

Applicants Response to Examiner's Comments

Examiner's Response to Declaration under CFR 1.131

Examiner objects to the Declaration under CFR 1.131 as filed on January 22, 2007 as failing to present evidence sufficient to the conception of the invention by demonstrative evidence.

5 Applicant respectfully submits that the enclosed Exhibits A through H evidence that the method of the present invention was conceived and reduced to practice without reference to prior art, and that Applicant produced working models of present invention prior to calendar year 2002.

10 To support Applicant's attestation that the method of the present invention was reduced to practice prior to the filing date of USPTO Patent Application 20030154197, Millet et al. of August 14, 2003, Applicant includes herein the following eight Exhibits A-G that were produced at the time during the initial implementation phases of the invention:

- A. ExtraView Objects Definition, authored no later than April 27, 2001;
- B. ExtraView_Schema_3.1.2.1.doc, authored no later than September 15, 2001;
- 15 C. ExtraView Administrator's Guide Version 3.1.2.1, authored no later than July 24, 2001;
- D. Problem_UDF.java, a Java software code for generating SQL for accessing a UDF, authored no later than December 30, 2002;
- E. Report.java, containing a a Java software code for generating SQL for
20 accessing a UDF, authored no later than December 19, 2002;
- F. ExtraView.pdf (diagram of ExtraView database schema), authored no later than March 11, 2002;
- G. Internal electronic correspondence regarding SQL generation for UDF access, with a subject heading of "mo betta SQL statement", emailed on October 10,
25 2002; and
- H. Internal electronic correspondence regarding SQL generation for UDF access, a with subject heading of "From eternity to 16 seconds", emailed on May 8, 2002.

Applicant further attests to the following timeline of development and application of the present invention:

- 1999-2000: implementation and delivery of PL/SQL version of ExtraView, including user-defined fields (UDF's);
- 2000-2001: re-engineering of the PL/SQL into Java code for the first version of ExtraView (Java) in December, 2001;
- 2001-2002: completion and enhancement of the UDF capabilities in subsequent ExtraView versions; and
- 2007:- version 5.2 of ExtraView.

More specifically, a UDF implementation according to the method of the present invention comprises the following aspects:

1. A database schema that allows users to add unlimited numbers of UDF's to be added to problem records.
2. Software logic to manage the UDF's, including creation, deletion, and modification of UDF attributes.
3. Graphical user interface using a Web browser, for accessing the functionality of UDF's; for example, administration of UDF's and retrieval or definition of UDF values for UDF instances in problem records.
4. Reporting functionality to provide searching/filtering by UDF and rendering of UDF values and titles to output devices such as web browser, Excel spreadsheets, or text documents

Applicant maintains that a lack of any of these crucial aspects of the method of the present invention would render application of the UDF concept much less valuable:

Applicant respectfully submits that the exhibits provided herein indicate how to use the UDF's evolved in accordance with the method of the present invention as claimed in Applicant's ExtraView product prior to 2002.

The software and these documents were used by real customers in companies to solve business problems with UDF's in the ExtraView product. This is *prima facie* evidence that an implementation existed and included the four specific aspects that are crucial to the invention.

EXHIBIT A: Internal UDF Software Object Definition

The following is taken from the enclosed Exhibit A, a written document entitled "ExtraView Objects Definition", dated April 27, 2001. Applicant notes that many of the data structures necessary to implement the claims of the invention are described therein: Attention is respectfully drawn to page 3 wherein the following statement was made.

(Beginning of insertion of page 3 of Exhibit A.)

- Titled Object: an abstract base class
 - ID
 - Name
 - Title
 - Sort sequence (within TitledObjects with same ID)
- User-Defined Field (aka UDF): is a Titled Object & has:
 - Data Type (Text Field, Text Area, Log Area, Number, Date, List, or URL)
 - Flags:
 - required/optional,
 - display-as-url/display-normal,
 - select-on-reports/no-select-on-reports
 - Help Text
 - Help URL
 - UDF Values (1:0..n relationship to UDF Value)
- UDF Value has:
 - Problem Id
 - UDF Id
 - Value
 - Value Number
 - Value Date
 - UDF List Membership (1:1 relationship to UDF List)
- UDF List : has
 - Title
 - Owning User
 - (1:0..n relationship to UDF)

(End of insertion of page 3 of Exhibit A.)

EXHIBIT F: Schema for UDF Tables

Also included as Exhibit F is a rendition of the schema used for the 4.0.1 release of an ExtraView.TM software. The Exhibit F document is dated March 11 2002, but it clearly documents the schema that was used earlier for the version released in December, 2001. The particular tables of importance to UDF's are:

- UDF (Page 5);
- PROBLEM_UDF (Page 5);
- UDF_LIST (Page 5);
- PROBLEM_TEXT (Page 5); and
- PROBLEM_UDF_HIST (Page 2)

More particularly:

- The UDF is the primary dictionary table for looking up UDF characteristics, such as title, etc.;
- PROBLEM_UDF is where the UDF values are stored, and values may take on different data types and therefore there are multiple value columns;
- UDF_LIST contains the list values for enumerated UDF's;
- PROBLEM_TEXT contains the text for the long text-valued UDF's.; and
- PROBLEM_UDF_HIST contains history for the UDF values in PROBLEM_UDF.

EXHIBIT B: Problem_UDF

The following description is copied from pages 59 & 60 enclosed Exhibit B, a written document entitled "ExtraView_Schema_3.1.2.1", dated September 15, 2001.

(Beginning of insertion of pages 59 & 60 of Exhibit B.)

PROBLEM_UDF

Intersection data associating issues to User Defined Fields, including text, numeric, or date value.

Column Name	Title	Format	Text
-------------	-------	--------	------

CREATED_BY_USER		VARCHAR2(30)	The user who created this problem User Defined Field.
DATE_CREATED		DATE	The date when this problem User Defined Field was created.
LAST_DATE_UPDATED		DATE	The date when this User Defined Field was last updated.
LAST_UPDATED_BY_USER		VARCHAR2(30)	The user that last updated this problem User Defined Field.
PROBLEM_ID		NUMBER	The identifier of the issue to which this problem User Defined Field belongs.
UDF_ID		NUMBER	The identifier of the User Defined Field to which this problem User Defined Field belongs.
UDF_LIST_ID		NUMBER	The identifier of the User Defined Field list to which this problem User Defined Field

			belongs.
VALUE		VARCHAR2(256)	The text value for this User Defined Field for an issue.
VALUE_DATE		DATE	The date value for this User Defined Field for an issue.
VALUE_NUMBER		NUMBER	The numeric value for this User Defined Field for an issue.

(End of insertion of pages 59 & 60 of Exhibit B.)

EXHIBIT C: Graphical User Interface

5 There are two major aspects to the graphical user interface aspects of UDF's: administration of UDF's and usage of the UDF in adding or editing problem records. From the Exhibit C, a printed document entitled "ExtraviewAdministration Guide Version 3.1.2.1", the screen shot (as found on page 15 therein) illustrates a method of adding a new UDF. The insertion of this screen shot of page 15 of Exhibit C begins on the following page of this

10 communication.

Add entry to the User Defined Fields table

Name

Title

Data table name

Column name in table

Data Type ☒ TEXTFIELD ☐ TEXTAREA ☐ LOGAREA ☐ NUMBER ☐ DATE ☐ LIST ☐ TAB ☐ URL

Required field ☐ Yes ☒ No

Display as URL ☐ Yes ☒ No

URL

Allow selection on reports ☐ Yes ☒ No

Allow new entries to be added dynamically to list ☐ Yes ☒ No

Help text

Help URL

(End of insertion of page 15 of Exhibit C.)

- 5 Exhibit C discloses that once a UDF is created, the UDF is treated like any other field that can be added to a problem record through a layout editing administrative function. Then the screen for adding/editing might look something like the following screen shot (taken from a problem report in the ExtraView support site). Many of these fields are UDF's, including the "Description", "Comments", and "Resolution". An important aspect is that the user cannot
- 10 differentiate between a UDF and a native, inbuilt field.

EXHIBIT C: UDF Reporting

The Exhibits disclose that UDF values can be shown on reports with their associated titles. Reports can be generated for email and/or immediate HTML browsing, for example. The screen shot below, excerpted from page 79 of Exhibit C , shows an email report with values of several UDF's shown, e.g., "Platform", "OS" and "Clarify ID".

(Beginning of insertion of page 23 of Exhibit C.)

Subject: DEV - 12231

Date: Fri, 15 Jun 2001 16:53:09 -0700 (PDT)

From: "ExtraView" <extraview.ga@sesame.com>

To: "Robbie Lloyd" <rlloyd@sesame.com>

Edit	Bug #	Title		
	12231	test		
Product	Owner	Created	Last Modified	
NetOp	udele malabanan	08-MAR-2001	15-JUN-2001 16:51	
Category	Assigned To	View	Alt ID	Changed By
Hardware	rob lloyd	Private		RDB.LLOYD
Severity	Priority			
	m2			
Module				
PM				
A Component	Platforms	OS	Clarify ID	Customer
	800 TDM			
Test Case ID	Test Case Location			
Version Open	Status	Version Closed	Disposition	
1	Closed			
Description				
test				
Comments				
Workaround				
Release Notes				
Enclosures				
Attachments				

cc: Michelle Medina, rob lloyd, Robbie Lloyd. •

(End of insertion of page 23 of Exhibit C.)

EXHIBIT E: Report Logic

Access to UDF's for reports imposed an extra join on the problem_udf table. The Java code for generating SQL for accessing a UDF was as follows (from pages 63-65, Exhibit E, the 4.0 version of the document entitled "Report.java" module).

(Beginning of insertion of pages 63-65 of Exhibit E.)

```
5          //*****
          // * CREATING THE UDF JOINS
          //
10         fromTable.setLength(0);

          if ("UDF".equalsIgnoreCase(re.getDdeType())){
15             tableAlias = re.getDdeName(); //"UDF" + String.valueOf(udfCnt++);

             //This was for before we used views
             //-----
20             //tableAlias = "IU_" + ddeName;
             //tableListAlias = "UL_" + ddeName;
             //tableUdfAlias = "U_" + ddeName;

             distinctTables.put(tableAlias, thisDDE);

25             udfIndexHint.append(" " + Z.dbms.indexHintString(tableAlias,
"IX_ITEM_UDF1"));

             sortOrderInfo.setTableAlias(tableAlias);
             //sortOrderInfo.setTableListAlias(tableListAlias);
30             sortOrderInfo.setSortOrderField(sof);
             sortOrderInfo.setSortBy(tableAlias + ".VALUE");
             sortOrderInfo.setSortByRawVal(tableAlias + ".VALUE");

             fromTable.append("(select distinct "+re.getItem_UDF()+".item_id \"ITEM_ID\",
35 "); // need this for the bad_data problem

             if (this.hr){
                 fromTable.append(re.getItem_UDF()+". "+getHistKey(re.getItem_UDF(),true) +
40 " " +
                     getHistKey(re.getItem(),true)+" ", " );
             }

             if( "LIST".equalsIgnoreCase(re.getDdeDisplayType())
                || "POPUP".equalsIgnoreCase(re.getDdeDisplayType())
45                || "TAB".equalsIgnoreCase(re.getDdeDisplayType())){

                 fromTable.append(" udf_list.title TITLE, udf_list.udf_list_id VALUE,
udf_list.sort_seq SORT_SEQ " +
50                 " from udf, udf_list, "+re.getItem_UDF() +
                 " where udf.udf_id = "+re.getItem_UDF()+".udf_id "+
                 " and udf_list.udf_list_id = "+re.getItem_UDF()+".udf_list_id " +
                 " and udf.name = ? ) ");

                 sortOrderInfo.setSortBy(tableAlias + ".TITLE");
55                 sortOrderInfo.setSortByRawVal(tableAlias + ".VALUE");

             }
             else if( "DATE".equalsIgnoreCase(re.getDdeDisplayType()) ){
                 fromTable.append( re.getItem_UDF()+ ".value_date VALUE " +
```

```

        " from udf, "+re.getItem_UDF()+
        " where udf.udf_id = "+re.getItem_UDF()+".udf_id " +
        " and udf.name = ? ) " );
    }
5   else if( "NUMBER".equalsIgnoreCase(re.getDdeDisplayType()) ){
        fromTable.append( re.getItem_UDF()+".value_number VALUE " +
        " from udf, "+re.getItem_UDF()+
        " where udf.udf_id = "+re.getItem_UDF()+".udf_id " +
        " and udf.name = ? ) " );
10  }
    else if( "USER".equalsIgnoreCase(re.getDdeDisplayType()) ){
        sortOrderInfo.setSortBy( this.getFormattedNameForQuery(tableAlias,
        "security_user_id",
        "first_name",
15    "last_name") );

        fromTable.append(
            re.getItem_UDF()+ ".value VALUE, " +
            "su.first_name FIRST_NAME, su.last_name LAST_NAME, " +
20    " su.security_user_id SECURITY_USER_ID " +
            " from security_user su, udf, "+re.getItem_UDF()+
            " where udf.udf_id = "+re.getItem_UDF()+".udf_id " +
            " and su.security_user_id = " + re.getItem_UDF()+ ".value "+
            " and udf.name = ? ) " );
25    }
    else{

        fromTable.append( re.getItem_UDF()+ " .value VALUE " +
30    " from udf, "+re.getItem_UDF()+
        " where udf.udf_id = "+re.getItem_UDF()+".udf_id " +
        " and udf.name = ? ) " );
    }
    paramsSortOrder.add(re.getDdeName());
    typesSortOrder.add("STRING");

    // join in the UDF from table
    //-----
35    fromCnt++;
    jd.addAll(fromTable.toString(),
        tableAlias,
        jd.LEFT_JOIN,
        DRIVER_KEY,
        re.getDriverAlias(),
        DRIVER_KEY,
45    1);

    if (this.hr) {
        andSortOrder.append(" and " +tableAlias+ "." +
50    getHistKey(tableAlias,true) +
        Z.dbms.ojequals() + driverTable + "." +
        getHistKey(driverTable,false));
    }
55    soFields.put(String.valueOf(sof.getOrderRank()), sortOrderInfo);

```

This software code as documented in Exhibit E generates SQL similar to this example
 60 (for enumerated UDF values):

```
select * from item,
```



```

(select distinct item_udf.item_id ITEM_ID, udf_list.title TITLE, udf_list.udf_list_id
VALUE, udf_list.sort_seq SORT_SEQ
      from udf, udf_list, item_udf
      where udf.udf_id = item_udf.udf_id
      and udf_list.udf_list_id = item_udf.udf_list_id
      and udf.name = 'fred' ) udfView
where item.item_id = udfView.item_id;

```

(End of insertion of pages 63-65 of Exhibit E.)

10 The novel and inventive aspect of method of the present invention documented by Exhibit E is that the UDF values require additional, creative SQL generation that implements the access of the data structures described in the schema.

EXHIBIT D: Edit Retrieval of UDF Values

15 For the editing of an existing problem record, the existing UDF values must be retrieved for display in the EDIT screen. This is done by generating SQL and executing the SQL as documented in the software codes of Exhibit D, the document entitled "Problem_UDF.java" of December 19, 2002.

(Beginning of insertion of pages 7-9 of Exhibit D.)

```

20 /**
    * <p> Sets udf text values in hashmap. This is only used by presentation.
    * It sets 2 items in the HashMap. The first item is the 'TEXT' value in the problem_text
table
    * for udf_id associated with this problem id. The second is the value from the
    * problem_udf table that is associated with this problem id. This is either the
25 * value, value_date, value_number or udf_list_id depending on udf_type.</p>
    * @param String problemId
    * @param Connection dbconn
    * @param HashMap hash
    */
30 public static void setHash(SesameSession session, Connection dbconn, String problemId,
    HashMap hash,
    boolean includeLogAreas)
    throws Exception {

35     String udfType = null;
    String prevUdfId = null;
    String udfId = null;
    String udfName = null;

40     String text = null;

    StringBuffer udfText = new StringBuffer();

    if (!includeLogAreas) {
45         udfText.append(" select pt.udf_id, id_seq, text_seq, text, u.name ");
        udfText.append(" from udf u, data_dictionary dd, problem_text pt ");
        udfText.append(" where pt.problem_id = ? ");
        udfText.append(" and u.udf_id = pt.udf_id ");
        udfText.append(" and dd.name = u.name ");
50         udfText.append(" and dd.display_type <> 'LOGAREA' ");
        udfText.append(" order by 1, 2, 3 ");
    } else {

```

```

        udfText.append(" select pt.udf_id, id_seq, text_seq, text, u.name from problem_text
pt, udf u where problem_id = ? ");
        udfText.append(" and u.udf_id = pt.udf_id ");
        udfText.append(" order by 1, 2, 3 ");
5    }
    StringBuffer udfVal = new StringBuffer(" select pu.UDF_ID, VALUE, VALUE_NUMBER,
VALUE_DATE, pu.UDF_LIST_ID, TITLE, u.name ");

    udfVal.append(" from udf_list ul, problem_udf pu, udf u ");
    udfVal.append(" where problem_id = ? ");
10   udfVal.append("and ul.udf_list_id " + Z.dbms.ojequals() + " pu.udf_list_id");
    udfVal.append(" and pu.udf_id = u.udf_id ");

    PreparedStatement statement = dbconn.prepareStatement(udfText.toString());

15   statement.setString(1, problemId);
    try{
        // get long text udfs
        ResultSet result = statement.executeQuery();

20       StringBuffer sb = null;
        while ( result.next() ) {
            udfId = result.getString("UDF_ID");
            udfName = result.getString("NAME");
25         if(result.getInt("TEXT_SEQ") > 1){
                sb.setLength(sb.length() - Z.TEXT_OVERLAP);
                sb.append(result.getString("TEXT"));
            } else {
30                 sb = new StringBuffer(result.getString("TEXT"));
            }

            hash.put(udfName, sb.toString());
        }
    } catch (Exception e) {
35         ErrorWriter.write(e, ErrorWriter.LOG);
    } finally {
        statement.close();
    }

    PreparedStatement statement2 = dbconn.prepareStatement(udfVal.toString());

    statement2.setString(1, problemId);
    // get rest of udfs
45   try {
        ResultSet result2 = statement2.executeQuery();

        while (result2.next()) {
            udfId = result2.getString("UDF_ID");
            udfName = result2.getString("NAME");
50         udfType = Z.dictionary.getDisplayType(dbconn, udfName);
            String value = null;

            // get the value from the correct column, depending on the udf type
            if (DataDictionary.isListType(udfType)) {
55                 value = result2.getString("UDF_LIST_ID");

            } else if (udfType.equals("DATE")) {
                Timestamp dateVal = result2.getTimestamp("VALUE_DATE");
                Calendar date = Convert.toCalendar(dateVal);
60                 DbTime dbt = new DbTime(session, dbconn);

                if (date != null) {
                    dbt.setDbNow(date);
65                 value = dbt.getShortDate();
                }
            }
        }
    }

```

```

    } else if (udfType.equals("NUMBER")) {
        value = result2.getString("VALUE_NUMBER");
5
    } else {
        value = result2.getString("VALUE");
    }

10
    if (hash.get(udfName) != null) { // for multi-valued udfs, add to the array
        String[] udflist = null;

        if (hash.get(udfName) instanceof String[]) {
15
            String[] currlist = (String[]) hash.get(udfName);

            udflist = new String[currlist.length + 1];
            for (int i = 0; i < currlist.length; i++) {
                udflist[i] = currlist[i];
20
            }

            udflist[currlist.length] = value;

        } else {

25
            udflist = new String[2];
            udflist[0] = (String) hash.get(udfName);
            udflist[1] = value;
        }

30
        hash.put(udfName, udflist);

    } else { // not multi-valued, just put in hashmap
        hash.put(udfName, value);

35
        if ("DATE".equals(udfType)) {
            hash.put((udfName + "_EVDISPLAY"), value);

        } else {
40
            hash.put((udfName + "_EVDISPLAY"), result2.getString("TITLE"));
        }

    }

45
}
} catch (Exception e) {
    ErrorWriter.write(e, ErrorWriter.LOG);
} finally {
    statement2.close();
50
}
}

```

(End of insertion of pages 7-9 of Exhibit D.)

55 The novel and inventive aspect of the method of the present invention embodied by this code sequence of Exhibit D is that the retrieval of the values and display titles for the UDF requires creative SQL generation and retrieval of data in an efficient manner.

EXHIBIT H: Implementation Progress

Exhibit H comprises an email of May 8th, 2002 having a subject header of “from here to eternity in 16 seconds”, wherein some of the background work of the implementation of the method of the present invention is commented upon by the author of the email.

(Beginning of Exhibit H, “from eternity to 16 seconds”.)

From: Rick Banister [rick@sesame.com]
Sent: Wednesday, May 08, 2002 4:32 PM
To: Tom Strzemieczny; Maria Scharin; Carl Koppel; Michael Stebbins;
Steve Hoydic
Cc: dev@sesame.com
Subject: From eternity to 16 seconds

I've got the worse case query so far down to 14 second, which is to query on 5 problem fields, 1 udf, and sort by a UDF. The EXPLAIN PLAN shows ridiculously high cost, but the actual performance is very good, even with the SGA cache cleared by restarting the database.

There are two items, a DDL change to bring the values into the index for an index-only search, and changing the query to use FIRST_ROWS so it doesn't try a full index query of the PROBLEM_UDF table:

```
DROP INDEX IX_PROBLEM_UDF1;
CREATE INDEX IX_PROBLEM_UDF1
  ON PROBLEM_UDF (problem_id, udf_id, VALUE, VALUE_NUMBER, VALUE_DATE, UDF_LIST_ID
)
  TABLESPACE EXTRAVIEW_IDX;
ANALYZE INDEX IX_PROBLEM_UDF1 ESTIMATE STATISTICS SAMPLE 20 PERCENT; ANALYZE table PROBLEM_UDF
ESTIMATE STATISTICS SAMPLE 20 PERCENT;

select /*+ ORDERED first_rows
INDEX(PROBLEM1 IX_PROBLEM5)
INDEX(PROBLEM2 IX_PROBLEM6)
INDEX(PROBLEM3 IX_PROBLEM7)
INDEX(PROBLEM4 IX_PROBLEM8)
INDEX(PROBLEM5 IX_PROBLEM2)
index(pudf ix_problem_udf1)
*/ PROBLEM1.ID
from PROBLEM PROBLEM1, PROBLEM PROBLEM2, PROBLEM PROBLEM3 , PROBLEM
PROBLEM4 , PROBLEM PROBLEM5, problem_udf pudf, udf_list where PROBLEM1.PRIORITY in ('PRIORITY
2') and PROBLEM2.SEVERITY_LEVEL in ('MAJOR') and PROBLEM3.STATUS in ('OPEN') and
PROBLEM4.PRODUCT_NAME in ('EVJAVA') and PROBLEM5.ASSIGNED_TO in ('AIMEN') and PROBLEM2.ID =
PROBLEM1.ID and PROBLEM3.ID = PROBLEM2.ID and PROBLEM4.ID = PROBLEM3.ID and PROBLEM5.ID =
PROBLEM4.ID and pudf.problem_id(+) = problem5.id and nvl(pudf.udf_id(+),88) = 88 and
udf_list.udf_list_id(+) = pudf.udf_list_id and exists
(select 1
from problem_udf udf1
where udf1.problem_id = PROBLEM5.id
and udf1.udf_id = 88
and udf1.udf_list_id = 6231)
ORDER BY udf_list.title DESC
```

(End of Exhibit H, “from eternity to 16 seconds”.)

EXHIBIT G: Implementation Progress

In another evidence of implementation progress, Exhibit G comprises an email of October 10, 2002 having a subject header of “no betta SQL statement”, wherein more of the background work of the implementation of the method of the present invention is commented upon by the author of this email.

(Beginning of Exhibit G, "mo betta SQL statement".)

From: robert lange [rlange@sesame.com]
Sent: Thursday, October 10, 2002 11:59 AM
To: Dev@Sesame. Com
Cc: LeeAnn Pultz; Maria Scharin
Subject: mo betta SQL statement

I ran into some surprising results tuning a query that I want to pass on to anyone writing SQL -- at least for the Oracle platform. The change involved replacing two joins in the where clause with "pseudo tables" in the where clause. The first change decreased resources by a factor of 10,000 in the explain plan, but the resulting query still took 37 What was especially surprising is that in the final change, even though the explain plan didn't show significant difference, the execution speed changed dramatically: by a factor of 40 again.

the basic query involved reporting, where you have a list of ids, and the problem udf table. the first pass on the query looked like this (simplified for clarity):

```
select p.id, p2.id
from problem p, problem p2, problem_udf pu, udf u where p.id in (?, ?, ?, ?, ?, ?, ?, ?) and pu.udf_id =
u.udf_id and u.name = '?'
and p2.id in (select problem_id from problem_udf
              where udf_id = u.udf_id and u.value = p.id) ;
```

the key was to rewrite the code to put some of the where clause into the from clause. step 1 was:

```
select p.id, p2.id
from (select id from problem where id in (?, ?, ?, ?, ?, ?, ?, ?)) p,
     problem p2, problem_udf pu, udf u
where p.id in (?, ?, ?, ?, ?, ?, ?, ?)
and pu.udf_id = u.udf_id and u.name = '?'
and p2.id in (select problem_id from problem_udf
              where udf_id = u.udf_id and u.value = p.id) ;
```

this reduced resources per the explain plan by a factor of 10,000 or so, but the query still took 40 seconds to run for 5 ids. the second step was to apply the same strategy to eliminate the joins on the udf table:

```
select p.id, p2.id
from (select id from problem where id in (?, ?, ?, ?, ?, ?, ?, ?)) p,
     problem p2,
     (select problem_id, value from udf, problem_udf
        where udf.name = ? and problem_udf.udf_id = udf.udf_id) pu where pu.value = p.id and p2.id
= pu.problem_id ;
```

this version did not look different in the explain plan, but the execution time dropped to under a second.

I'm not sure that this is always more efficient, and it may have to do with the fact that the actual query was quite a bit more complex, but the approach clearly can have dramatic results.

enjoy,

robert

(End of Exhibit G, "mo betta SQL statement".)

Conclusion

Applicant respectfully submits that it is clear from the referenced documents that the design and implementation of the method of the present invention as claimed was

completed 2002. Indeed, the initial UDF implementations occurred in late 2000 and early 2001. Applicant therefore respectfully submits that date of invention of the method of the present invention predates the cited and related art disclosed in Millet et al., and that Claims 1-9 as currently amended are therefore allowable.

5 ***Examiner's Objections to Informalities of Amendment A***

Examiner objects to Amendment A as filed on January 22, 2007 for the informality of line numbers representing specific line numbers of the printed page running together with the amended Claims. Applicant replies that the Claims as amended herein do not present line numbers running together with Claim numbers.

10 ***Specification Amendment A***

Examiner objects to the amendment of the Specification as submitted on January 22, 2007. Applicant acknowledges that the amendment to the Specification as submitted on January 22, 2007 has not been entered.

Claim Objections

15 Examiner objects to Claim 6 for stating "the plurality of first datafields of each of the plurality of UDF's", as each UDF has only one "first datafield". Applicant has deleted the words "of each" from Claim 6 as currently amended, and therefore respectfully responds that Claim 6 is allowable.

20 Examiner objects to Claim 8 for missing the word "of" in front of the phrase "a plurality UDF's". Applicant has added the word "of" as suggested by Examiner in Claim 8 as currently amended. Applicant therefore respectfully responds that Claim 8 is allowable.

Claim Rejections – 35 USC § 112

25 Examiner rejects Claims 1-10 under 35 U.S.C. 101 as being directed to non-statutory subject matter. Examiner rejects Claims 1, 5, 6, 8 and 9 for use of the words "may be". Applicant has amended Claims 1, 5, 6, 8, and 9 and replaced the words "may be" with the word "is" in this communication and therefore respectfully responds that Claims 1, 5, 6, 8 and 9 are allowable.

Examiner rejects Claim 1 for lacking antecedent basis for the limitation of "the record" and the limitation "the table". Applicant responds that Claim 1 as currently amended recites "a

software record” and “a software table” in the preamble to Claim 1. Applicant respectfully submits that Claim 1 as currently amended is allowable.

Examiner rejects Claim 4 for lacking antecedent basis for the limitation of “the name”. Applicant responds that Claim 4 as currently amended reads “a name”, and respectfully submits that Claim 4 as currently amended is allowable.

Applicant cancels Claim 10.

Claim Rejections – 35 USC § 101

Examiner rejects Claims 1-10 under 35 U.S.C. 101 as being directed to non-statutory subject matter.

Examiner rejects Claims 1-4 and 9 because the term “computer-readable medium” is not limited to tangible medium in Paragraph 0102 of the Specification as originally filed, but rather includes an acoustic or light wave as a form of “computer-readable medium”. Applicant responds that Claims 1-9 have been amended to method claims and that the objection to the scope of definition of the term “computer-readable medium” as an article of manufacture is therefore moot.

Applicant respectfully therefore submits that the Claims 1-9 as currently amended satisfy the requirements of 35 U.S.C. 101 and are allowable.

Claim Rejections – 35 USC § 102

Examiner rejects Claims 1-10 under 35 U.S.C. 102 paragraph (e) as being anticipated by a Patent Application by Millet, et al. (Publication 2003/0154197).

Applicant respectfully directs Examiner’s attention to the accompanying Declaration under 37 C.F.R. § 1.131, wherein Applicant submits an attestation and supporting documentation of Exhibits A through G that Applicant’s data of invention is antecedent to that of Millet, et al., and that Millet et al. is therefore not Prior Art as required under 35 U.S.C. 102 paragraph (e).

Examiner rejects claims 1-10 under 35 U.S.C. 102(e) as being anticipated by Millet et al. (Publication No US 2003/0154197).

Examiner rejects claim 1 as being anticipated by Millet et al. , and that Millet et al. teach: "A computer-readable medium having stored thereon a computer-readable program code comprising a sequence of instructions which, when executed by a computer, cause the computer

to perform steps" (as per Abstract and [0040]) comprising:

"forming a first user-defined data field structure, or first "UDF" (as per Abstract, [0040], [0048] and Fig. 13 each of the records in the "Custom Field Values" data table is equivalent to Applicant's "UDF"), the first UDF comprising:

5 "an record identifier datafield" (as per [0042], [0044], [0048], [0054] and Fig. 13 for "Row 10" datafield); "an UDF identifier datafield" (as per [0048] and Fig. 13 for "Field 10" datafield); "a first datafield" (as per [0048], Fig. 13 and Fig. 15 wherein "Value" or FieldValue" datafield is equivalent to Applicant's "first datafield"); "storing a record identifier in the record identifier datafield" (as per Fig. 13); "storing a UDF identifier in the UDF identifier datafield" (as
10 per Fig. 13); and "storing an additional information in the first datafield, whereby the first datafield is associated with the record and the additional information stored in the first datafield may be associated with the record and without modification of the table" (as per Abstract, [0048] and Fig. 13).

Applicant replies that the reference of Millet et al. as cited by the Examiner is not Prior
15 Art under 35 U.S.C. 102 and that Claim 1 is therefore allowable.

Examiner rejects claim 2 based on arguments given above for rejected claim 1 and is also additionally rejecting this claim for the following:

Examiner holds that Millet et al. teach:

"forming a metadata, the metadata comprising a classification of data type, the
20 classification of data type distinguishing the data type of the additional information stored in the first datafield; and associating the metadata with the first UDF" (as per [0048], [0056] and Fig. 10 wherein attribute information is equivalent to Applicant's "metadata", the type of data in the field is equivalent to Applicant's "classification of data type"; also see [0041] and [0073]).

Applicant respectfully replies that Claim 2 depends from the independent and allowable
25 Claim 1, and is therefore allowable.

Examiner rejects claim 3 based on arguments given above for rejected claim 1 and is also additionally rejecting this claim for the following:

Examiner holds that Millet et al. teach:

"forming a metadata, the metadata associated with the first UDF, and the metadata

comprising a name, the name associated with the first UDF and the name for use in software operations accessing the first UDF; and

associated the metadata with the first UDF" (see Abstract, [0048], [0056] and Fig. 10 wherein field attribute information store in "Custom Fields" table is equivalent to Applicant's "metadata

5 associated with the first UDF", and field name such as "Memo" is associated with a custom field or UDF as illustrated in Applicant's claim language; also see [0073]).

Applicant respectfully replies that Claim 3 depends from the independent and allowable Claim 1, and is therefore allowable.

Examiner rejects claim 4 based on arguments given above for rejected claim 1

10 and is also additionally rejecting this claim for the following:

Examiner holds that Millet et al. teach:

"forming a metadata, the metadata comprising a title, the title associated with the first UDF and the name for use in a visual display of the additional information of the first UDF; and associating the metadata with the first UDF (as per [0056] wherein field attribute information in "Custom Fields" table is equivalent to Applicant's "metadata associated with the first UDF", and 15 "text associated with that field" is equivalent to title as illustrated in Applicant's claim language).

Applicant respectfully replies that Claim 4 depends from the independent and allowable Claim 1, and is therefore allowable.

Examiner Rejects claim 5 based on arguments given above for rejected claim 1

20 and is also additionally rejecting this claim for the following:

Examiner holds that Millet et al. teach:

"forming a class plurality of UDF's and wherein the first datafield comprises a class identifier of the class plurality of UDF's" (as per [0048], Fig. 12 and Fig.15 wherein set of custom fields associated with each database table within the RDBMS is equivalent to Applicant's 25 "a class plurality of UDF's", and "TableID" or "ValueID" is equivalent to Applicant's "class identifier"), and each UDF of the class plurality includes:

"the class identifier" (as per Fig. 10-12 wherein "TableID" is equivalent to Applicant's claim language);

"a unique identifier of the UDF of the class plurality of UDF's" (as per Fig. 10 wherein

"FieldID" is equivalent to Applicant's claim language); and

"a datafield, whereby each datafield of the class plurality of UDF's may be associated with the first UDF and therefrom associated with the record, and information may be stored in the plurality of data fields of the class plurality of UDF's and associated with the first UDF's, and therefrom the information of the plurality of data fields of the class plurality of UDF's may be associated with the record and without modification of the table (as per Fig. 10, and Fig 13 wherein records of table in Fig. 10 is equivalent to Applicant's "the class plurality of UDF's", each record of table in Fig. 13 is equivalent to Applicant's "first UDF", the "FieldID" in both table indicates the association between two table or the association between the class plurality of UDF's and the first UDF as in Applicant's claim language, the "RowID" in table of Fig. 13 indicates the associate between that table and the database table.

Examiner maintains that Millet et al. teaches the association between UDF and class plurality of UDF's with records in the database table as illustrated in Applicant's claim language; also see [0057]).

Applicant respectfully replies that Claim 5 depends from the independent and allowable Claim 1, and is therefore allowable.

Examiner rejects claim 6 based on arguments given above for rejected claim 1 and is also additionally rejecting this claim for the following:

Examiner holds that Millet et al. teach:

"forming a plurality of UDF's" (as per Fig. 15 and [0057] wherein "Custom Field Values" table is equivalent to Applicant's "data structure" and its records is equivalent to Applicant's "a plurality of UDF's");

"storing an identifier of the first UDF In the record identifier datafield of each of the plurality of UDF" (as per Fig. 15 wherein "FieldID" is equivalent to Applicant's claim language);

"storing a unique identifier in the record identifier datafield of each of the plurality of UDF's" (as per Fig. 5 wherein "ValueID" is equivalent to Applicant's claim language); and

"storing information in each of the first datafields of each of the plurality of UDF's.

Examiner asserts that the plurality of first datafields of each of the plurality of UDF's are associated with the first UDF and information may be stored in the plurality of datafields and associated with the first UDF and therefrom the information of the plurality of datafields may be associated with the record and without modification of the table" (as per Fig. 5 and [0057]-
5 [0058] wherein "RecordID" is equivalent to "datafield" as illustrated in Applicant's claim language since "RecordID" is a primary/foreign key which allows connecting record to another record which includes plurality of datafields; also see [0046]).

Applicant respectfully replies that Claim 6 depends from the independent and allowable
10 Claim 1, and is therefore allowable.

Examiner rejects claim 7 based on arguments given above for rejected claim 1 and is also additionally rejecting this claim for the following:

Examiner holds that Millet et al. teach:

"to store a pointer in the record identifier datafield for use as the record identifier" (as per
15 Fig. 13, [0048] and [0054] wherein "RowID" is equivalent to Applicant's "a pointer" and "the record identifier" because it points to location of the record in the table).

Applicant respectfully replies that Claim 7 depends from the independent and allowable Claim 1, and is therefore allowable.

Examiner Rejects claim 8 based on arguments given above for rejected claim 1 and is
20 also additionally rejecting this claim for the following:

Millet et al. teach:

"forming a plurality of UDF's, each UDF associated with a same record stored in a table" (Fig. 15 and [0057] wherein "Custom Field Values" table is equivalent to Applicant's "data structure" and its records is equivalent to Applicant's "a plurality of UDF's"), "whereby the
25 plurality of first datafields are associated with the same record and information may be stored in the plurality of datafields and associated with the first UDF and therefrom the information of the plurality of datafields may be associated with the same record and without modification of the table" (as per Fig. 5 and [0057]-[0058] wherein "ValueID" is equivalent to "datafield" as illustrated in Applicant's claim language since "ValueID" is a

primary/foreign key which allows connecting record to another record which includes plurality of datafields; also see [0046]).

Applicant respectfully replies that Claim 8 depends from the independent and allowable Claim 1, and is therefore allowable.

5 Examiner rejects claim 9 based on arguments given above for rejected claim 1 and is also additionally rejecting this claim for the following:

Examiner holds that Millet et al. teach:

"forming a data structure having a record, a list and a list user-defined field, or List
"UDF", the List UDF relatable to the record" (as per [0041], [0048] and Fig. 15 wherein a table
10 is a list of records, and any custom field associated with the table is equivalent to Applicant's
"list user-defined field"), and the List UDF comprising: "an identifier of the List UDF" (as per
Fig. 10 wherein "FieldID" is equivalent to Applicant's claim language); "an identifier of the List"
(as per Fig. 10 wherein "TableID" is equivalent to Applicant's "claim language"); and "a data
address of the List, whereby an information stored at the data address of the list is associated
15 with the List UDF and the information may be stored or modified at the data address of the list
and the information may be associated with the record and without modification of the table" (as
per [0073] and [0074] wherein the second values table as disclosed is equivalent to the List and
the disclosure of retrieval of information from the table implies the inclusion of some data
address to access table from its storage).

20 Applicant respectfully replies that Claim 9 depends from the independent and allowable Claim 1, and is therefore allowable.

Examiner rejects claim 10 as being anticipated by Millet et al. teach:

"A computer system" (as per Abstract) comprising:

"a controller" (as per [0040]); "a memory, the memory communicatively coupled with the
25 controller and the memory storing a software database and a software database manager" (as per
[0040] for database and database application); "a software database having data organized into a
table of records" (as per [0041]); "a user-defined field stored in the memory for associating a
datum with a record of the table, the user defined field having a UDF identifier and a record
identifier" (as per [0040], [0048], [0073] and Fig. 13 and 15 wherein "FieldID" is equivalent to

Applicant's "UDF identifier", and "RowID" or "RecordID" is equivalent to Applicant's "record identifier");

"a metadata stored in the memory and associated with the user-defined field and the metadata specifying the data type of the datum" (as per [0056] and Fig. 10 wherein field attribute information is equivalent to metadata as illustrated in Applicant's claim language); and

"the database manager software program for directing the controller to merge the user-defined field with the record to associate the datum of the user-defined field with the record of the table" (as per [0040] and [0061] wherein the application allowing user to add data column as necessary as disclosed is equivalent to Applicant's "database manager software program").

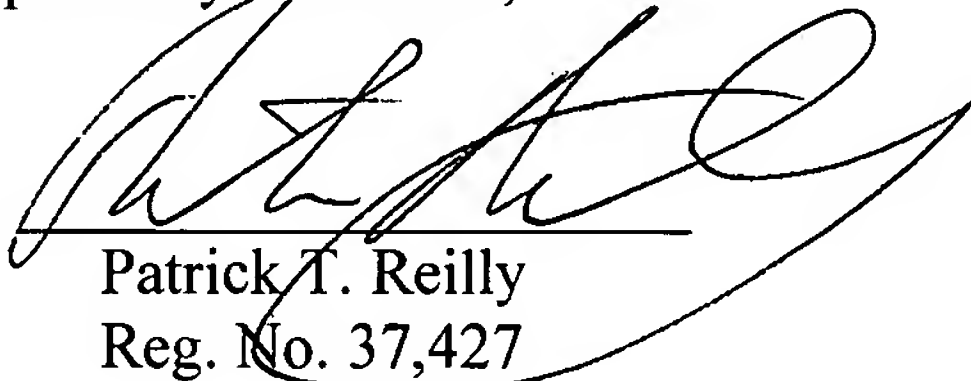
Applicant replies that Claim 10 is cancelled, while reserving the right to submit Claim 10 in a Continuation Application, and that Examiner's arguments are therefore moot.

In summation, Applicant respectfully submits that the Examiner's objections to the Claims have been fully resolved by the Claims 1-9 as currently amended. Applicant further respectfully submits that the Examiner's rejections of the Claims have been fully traversed by scope and recitations of the Claims 1-9 as currently amended, and that the Claims 1-9 are therefore allowable.

If any matters can be resolved by telephone, Applicant requests that the Patent and Trademark Office call the Applicant at the telephone number listed below.

Respectfully submitted,

By:


Patrick T. Reilly
Reg. No. 37,427

Patrick Reilly
Patent Attorney
Box 7218
Santa Cruz, CA 95061-7218
(831) 332-7127

EXHIBIT E

149 PAGES

```
Report.java
package com.extraview.applogic.report;

import java.sql.*;
import java.io.*;
import java.util.*;
import com.extraview.applogic.*;
import com.extraview.applogic.admin.*;
import com.extraview.applogic.layout.*;
import com.extraview.applogic.security.SecurityUser;
import com.extraview.usercustom.*;
import com.extraview.presentation.Display;
import com.extraview.presentation.chart.*;
import com.extraview.util.*;
import com.extraview.common.*;
import com.extraview.dbms.*;
import com.stevesoft.pat.*;
import com.sesame.log.Log;
import com.sesame.misc.ErrorWriter;
import com.extraview.applogic.security.SecurityPermission;

public class Report extends TitledObject implements Transactional, Cacheable,
Serializable, Sequenced {

// Transactional
    private boolean modifiedData = false;
    private boolean newData = false;
    private boolean deletedData = false;
    private transient Connection conn = null;

// Audit
    private String createdBy = null;
    private String updatedBy = null;
    private java.sql.Timestamp createdOn = null;
    private java.sql.Timestamp updatedOn = null;

// Local
    private boolean isSummaryReport = false;
    private boolean isStatusOnDate = false;
    private boolean isProductOnDate = false;
    private boolean isReleaseOnDate = false;
    private boolean isStatusByDate = false;
    private boolean isProductByDate = false;
    private boolean isReleaseByDate = false;
    private boolean isChart = false;
    private int timeInterval = 4;

    private boolean hr = false;
    private boolean hasCalculatedFields = false;
    private boolean isCaseInsensitive = false;
    private boolean unlimitedResults = false;
    private int reportId = 0;
    private int filterGroupId = 0;
    private int layoutId = 0;
    private int sortOrderId = 0;
    private int reportGroupId = 0;
    private int chartId = 0;
    private String outputTypeName = null;
    private String reportTypeName = null;
    private String securityGroupId = null;
    private String securityUserId = null;
    private String driverTable = "ITEM";
```



```

Report.java
private String reportListURL = null;

// Reporting Objects
private FilterGroup fg = null;
private ReportChart reportChart = null;
private Layout l = null;
private SortOrder so = null;
private ArrayList calculatedFields = new ArrayList();
private String concatHint = Z.dbms.concatHint() + " ";
private String concatHintString = Z.dbms.concatHintString();
private HashMap suppressedMultiReleaseFilters = new HashMap(); // reback
thing- they only want to see the child releases they filter on in the report results

// Constants
private static final int MAX_PERSIST_LIST_SIZE = 5000; /// this MUST BE A
MULTIPLE OF 20 -- the INTERNAL PAGE SIZE;
private static final String _TEXT = "&%TEXT";
private static final String _TIMESTAMP = "&%TIMESTAMP";
private static final String _USER = "&%USER";
private static final String q = "\"";
private static final String sufx0 = "";
private static final String sufx1 = "_1";
private static final String sufx2 = "_2";
private static final String sufx3 = "_3";
private static final String tagPrefix = ""; // "_TAG_";
private static final String tagSuffix = ""; // "_";
public static final String PROBLEM_KEY_ALIAS = "ID"; // this is the alias
included in every select query for problem_id
public static final String RELEASE_KEY_ALIAS = "PROBLEM_RELEASE_ID"; // this is
the alias included in every select query for problem_release_id
public static final String PROBLEM_KEY = "ITEM_ID"; // this is the key included
in every select query for problem_id
public static final String RELEASE_KEY = "PROBLEM_RELEASE_ID"; // this is the
key included in every select query for problem_release_id
public static final String DRIVER_KEY = "ITEM_ID";

private String emptyStringsSpacer = "&nbsp;";

// Used when querying - internal block stuff/ paging / pageblock
private boolean isNewPersist = false; // determines if you start fresh
getList() or append to last pageblock
private boolean startedPageBlockCalc = false; // flag used for setting the
starting counter for the internal blocks within a pageblock
private boolean searchAttach = false; // whether to search Blob attachments
private int maxRows = 0; // the maximum number of rows allowed for an unlimited
search for non BYPASS users
private boolean filterChildValue = false; // used for reback originally to
only show releases they filter on
private boolean checkMaxRows = false;
private final int INTERNAL_BLOCK_SIZE = 20; // the number of records to be
retrieved/processed in memory at a time
private final int PAGES_PER_PAGEBLOCK = 10; // the number of pages to be
hyperlinked at the bottom of the page
private ValidationList resultBlock = null;
private Enumeration resultEnum = null;
private int resultCount = 0;
private int totalAttachmentSize = 0;
private int unlimitedBlock = 0;
private int elementStartAt = 0;
private int pageBlock = 0; // the number of the block of 10 pages (ie 1st 10
pgs = block 1)
private ArrayList pageKeys = null;
private ArrayList blockKeys = new ArrayList();

```

Report.java

```

private int resultPerPage = 0;
private int internalBlockStartElement = 0; // the element to start processing
on for the next internal block

private HashMap dataAlias = new HashMap();
private HashMap multiLookups = null; // used to track records with multiple
lookup values
private HashMap prSorts = new HashMap(); //tracks the sorts that have been added
for child release records
private HashMap prFilters = new HashMap(); //tracks the Filters that have been
added for child release records

// Regex stuff
private String esc = "(?e=~)";
// Don't put regex'es in here: they are not serializable.

/**
 * Report()
 *
 * Constructor used to instantiante an new Report object to populate.
 * Gets initial ID from sequence.
 * To get an already existing object use getReference()
 */
public Report() throws Exception {
    super();
    newData = true;
    setOutputType("BROWSER");
}

/**
 * Report(String reportId)
 * @param reportId
 *
 * This constructor is called from get reference --- it is used given an
existing ID
 * -- note that newData is not set
 */
public Report(int reportId) {
    super();
    this.reportId = reportId;
    setOutputType("BROWSER");
}

/**
 * getSequence
 * <p>Ensures we return a number if we don't already have one.
 * @return String - an id number
 */
public String getSequence() {
    if (this.reportId < 1) {
        try {
            this.reportId =
Integer.parseInt(Sequence.getNewSequenceId("REPORT_SEQ"));
        } catch (Exception e) {
            ; // pass
        }
    }
    return String.valueOf(this.reportId);
}

public boolean isSummaryReport(){ return this.isSummaryReport; }
public boolean isStatusOnDate(){ return this.isStatusOnDate; }

```

```

Report.java
public boolean isProductOnDate(){ return this.isProductOnDate; }
public boolean isReleaseOnDate(){ return this.isReleaseOnDate; }
public boolean isStatusByDate(){ return this.isStatusByDate; }
public boolean isProductByDate(){ return this.isProductByDate; }
public boolean isReleaseByDate(){ return this.isReleaseByDate; }
public boolean isChart(){ return this.isChart; }

public int getTimeInterval(){ return this.timeInterval; }

public void setSummaryReport(boolean b){
    this.isSummaryReport = b;
}
public void setStatusOnDate(boolean b){
    this.isStatusOnDate = b;
    this.setIsChart(b);
}
public void setProductOnDate(boolean b){
    this.isProductOnDate = b;
    this.setIsChart(b);
}
public void setReleaseOnDate(boolean b){
    this.isReleaseOnDate = b;
    this.setIsChart(b);
}
public void setStatusByDate(boolean b){
    this.isStatusByDate = b;
    this.setIsChart(b);
}
public void setProductByDate(boolean b){
    this.isProductByDate = b;
    this.setIsChart(b);
}
public void setReleaseByDate(boolean b){
    this.isReleaseByDate = b;
    this.setIsChart(b);
}
public void setTimeInterval(int interval){
    this.timeInterval = interval;
}
public void setIsChart(boolean b){
    this.isChart = b;
}

private void cleanFlags(){
    this.setSummaryReport(false);
    this.setStatusOnDate(false);
    this.setStatusByDate(false);
    this.setProductOnDate(false);
    this.setProductByDate(false);
    this.setReleaseOnDate(false);
    this.setReleaseOnDate(false);
    this.setIsChart(false);
}

/**
 * populateObj
 */
public void populateObj(Map data) {}

/**
 * addSortOrder
 * @param SortOrder
 */

```

```

Report.java
public void addSortOrder(SortOrder sort) {
    this.so = (SortOrder) sort.clone();
    this.modifiedData = true;
}

/**
 * addLayout
 * @param Layout
 */
public void addLayout(Layout layout) {
    this.l = layout;
    this.modifiedData = true;
}

/**
 * getLayout
 * @returns Layout
 */
public Layout getLayout() {
    return this.l;
}

/**
 * addFilterGroup
 * @param FilterGroup
 */
public void addFilterGroup(FilterGroup filterGroup) {
    this.fg = (FilterGroup) filterGroup.clone();
    this.modifiedData = true;
}

/**
 * getFilterGroup
 * @returns FilterGroup
 */
public FilterGroup getFilterGroup() {
    return this.fg;
}

/**
 * getReportId
 * @returns filterGroupId
 */
public int getReportId() {
    return this.reportId;
}

/**
 * getLayoutId
 * @return layoutId
 */
public int getLayoutId() {
    return this.layoutId;
}

/**
 * setLayoutId
 * @param layoutId
 */
public void setLayoutId(int layoutId) {
    this.layoutId = layoutId;
    modifiedData = true;
}

```

Report.java

```
/**
 * getResultCount
 * @return ResultCount
 */
public int getResultCount() {
    return this.resultCount;
}

/**
 * getSortOrderId
 * @return sortOrderId
 */
public int getSortOrderId() {
    return this.sortOrderId;
}

/**
 * setsortOrderId
 * @param sortOrderId
 */
public void setSortOrderId(int sortOrderId) {
    this.sortOrderId = sortOrderId;
    modifiedData = true;
}

/**
 * getSortOrderId
 * @return sortOrderId
 */
public SortOrder getSortOrder() {
    return this.so;
}

/**
 * getFilterGroupId
 * @return filterGroupId
 */
public int getFilterGroupId() {
    return this.filterGroupId;
}

/**
 * setFilterGroupId
 * @param filterGroupId
 */
public void setFilterGroupId(int filterGroupId) {
    this.filterGroupId = filterGroupId;
    modifiedData = true;
}

/**
 * getOutputTypeName
 * @return outputTypeName
 */
public String getOutputType() {
    return this.outputTypeName;
}

/**
 * setOutputType
 * @param outputTypeName
 */
```

```

Report.java
public void setOutputType(String outputTypeName) {
    this.outputTypeName = outputTypeName;
    if ("BROWSER".equalsIgnoreCase(outputTypeName)) emptyStringSpacer =
"&nbsp;";
    else if ("TEXT".equalsIgnoreCase(outputTypeName)) emptyStringSpacer = "";
    else if ("MS_EXCEL".equalsIgnoreCase(outputTypeName)) emptyStringSpacer =
"";
    else if ("MS_WORD".equalsIgnoreCase(outputTypeName)) emptyStringSpacer =
"&nbsp;";
}

/**
 * isHTMLOutputType
 * @param outputTypeName
 */
public boolean isHTMLOutputType() {
    if ("BROWSER".equalsIgnoreCase(outputTypeName)
        || "MS_WORD".equalsIgnoreCase(outputTypeName)
        || "MS_EXCEL".equalsIgnoreCase(outputTypeName)) {
        return true;
    } else return false;
}

public boolean isTextOutputType() {
    return "TEXT".equalsIgnoreCase(this.outputTypeName);
}

/**
 * getReportType
 * @return outputTypeName
 */
public String getReportType() {
    return (this.reportTypeName == null ? "" : this.reportTypeName);
}

/**
 * setCaseInsensitiveSearch
 * @param isCaseInsensitive
 */
public void setCaseInsensitiveSearch( boolean isCaseInsensitive ) {
    this.isCaseInsensitive = isCaseInsensitive;
}

/**
 * setReportType
 * @param ReportTypeName
 */
public void setReportType(String reportTypeName) {
    this.reportTypeName = reportTypeName;

    //Z.probe("RPT: reportTypeName =" +this.reportTypeName );
    if (this.reportTypeName.indexOf("HISTORY") > -1) {
        this.driverTable = "ITEM_HIST";
        this.hr = true;
    } else {
        this.driverTable = "ITEM";
        this.hr = false;
    }
}

/**

```

Report.java

```

    *
    **/
public boolean isHistoryReport() {
    return this.hr;
}

/**
 *
 **/
public void setSearchAttach(boolean searchAttachment) {
    this.searchAttach = searchAttachment;
}

/**
 *
 **/
public boolean getSearchAttach() {
    return this.searchAttach;
}

/**
 *
 **/
public boolean hasCalculatedFields() {
    if (this.calculatedFields.size() > 0)
        this.hasCalculatedFields = true;
    else this.hasCalculatedFields = false;
    return this.hasCalculatedFields;
}

/**
 * setResultPerPage
 * @param int resultPerPage
 **/
public void setResultPerPage(int resultPerPage) {
    this.resultPerPage = resultPerPage;
    if (this.resultPerPage == -1) this.unlimitedResults = true;
}

/**
 * getResultPerPage
 * @return int
 **/
public int getResultPerPage() {
    return this.resultPerPage;
}

/**
 * setReportListURL
 * @param ReportListURL
 **/
public void setReportListURL(String reportListURL) {
    this.reportListURL = reportListURL;
    modifiedData = true;
}

/**
 * getReportListURL
 * @return reportListURL
 **/
public String getReportListURL() {
    return this.reportListURL;
}

```



```
}

/**
 * getCalculatedFields
 * @return calculated fields
 */
public ArrayList getCalculatedFields() {
    return this.calculatedFields;
}

/**
 * getChartId
 * @return chart id
 */
public int getChartId() {
    return this.chartId;
}

/**
 * getReportChart
 * @return reportChart
 */
public ReportChart getReportChart() {
    return this.reportChart;
}

/**
 * getPresentationChart
 * @return chart
 */
public Chart getPresentationChart() {
    return this.reportChart.getChart();
}

/**
 * setLastUpdatedByUser
 * @param userId
 */
public void setLastUpdatedByUser(String userId) {
    this.updatedBy = userId;
    modifiedData = true;
}

/**
 * getSecurityUserId
 * @return securityUserId
 */
public String getSecurityUserId() {
    return this.securityUserId;
}

/**
 * setSecurityUserId
 * @param securityUserId
 */
public void setSecurityUserId(String securityUserId) {
    this.securityUserId = securityUserId;
    modifiedData = true;
}

/**
 * getSecurityGroupId
```

```

    * @return securityGroupId
    */
    public String getSecurityGroupId() {
        return this.securityGroupId;
    }

    /**
     * setSecurityGroupId
     * @param securityGroupId
     */
    public void setSecurityGroupId(String securityGroupId) {
        this.securityGroupId = securityGroupId;
        modifiedData = true;
    }

    /**
     *
     */
    public void setName(String value) {
        this.name = value;
        modifiedData = true;
    }

    /**
     *
     */
    public void setTitle(String value) {
        this.title = value;
        modifiedData = true;
    }

    /**
     * setFilterChildValue
     * @param boolean filterChildVale
     */
    public void setFilterChildValue(boolean filterChild){
        this.filterChildValue = filterChild;
    }

    /**
     * getReference( Connection conn, int reportId )
     * @param conn
     * @param reportId
     * @return Report
     * @throws InvalidReportException
     *
     * creates a static reference of Report for a given reportId
     */
    public static Report getReference(Connection conn, int reportId) throws
InvalidReportException, Exception {
        Report r = null;

        // check if we have a connection
        if (conn == null) throw new Exception("Null connection");
        if (reportId == 0) throw new InvalidReportException("Report Id is null");

        String query = "select          REPORT_ID, " +
            "          LAYOUT_ID, " +
            "          SORT_ORDER_ID, " +
            "          FILTER_GROUP_ID, " +
            "          OUTPUT_TYPE_NAME, " +
            "          REPORT_TYPE_NAME, " +
            "          REPORT_GROUP_ID, " +

```

```

        Report.java
        "    TITLE, " +
        "    DESCRIPTION, " +
        "    SECURITY_GROUP_ID, " +
        "    SECURITY_USER_ID, " +
        "    DATE_CREATED, " +
        "    LAST_DATE_UPDATED, " +
        "    LAST_UPDATED_BY_USER, " +
        "    CREATED_BY_USER, " +
        "    TITLE_MAP_KEY, " +
        "    REPORT_LINK_URL " +
        "from report " +
        "where report_id = ? ";

String cfQuery = "select CALCULATED_FIELD_ID, " +
    " REPORT_ID, " +
    " NAME, " +
    " CALCULATION, " +
    " ORDER_RANK " +
    " from CALCULATED_FIELD " +
    " where report_id = ? " +
    " order by order_rank ";

PreparedStatement statement = null;
PreparedStatement cfStatement = null;

try {

    statement = conn.prepareStatement(query);
    statement.setInt(1, reportId);
    ResultSet result = statement.executeQuery();

    // Get the information associated with this report_id
    //-----
    if (result.next()) {
        r = new Report(reportId);
        r.reportId = result.getInt("REPORT_ID");
        r.layoutId = result.getInt("LAYOUT_ID");
        r.sortOrderId = result.getInt("SORT_ORDER_ID");
        r.filterGroupId = result.getInt("FILTER_GROUP_ID");
        r.outputTypeName = result.getString("OUTPUT_TYPE_NAME");
        r.setReportType(result.getString("REPORT_TYPE_NAME"));
        r.reportGroupId = result.getInt("REPORT_GROUP_ID");
        r.name = result.getString("TITLE");
        r.title = result.getString("DESCRIPTION");
        r.titleMapKey = result.getLong("TITLE_MAP_KEY");
        r.securityUserId = result.getString("SECURITY_USER_ID");
        r.securityGroupId = result.getString("SECURITY_GROUP_ID");
        r.createdOn = result.getTimestamp("DATE_CREATED");
        r.updatedOn = result.getTimestamp("LAST_DATE_UPDATED");
        r.updatedBy = result.getString("LAST_UPDATED_BY_USER");
        r.createdBy = result.getString("CREATED_BY_USER");
        r.reportListURL = result.getString("REPORT_LINK_URL");
    }
    else {
        throw new InvalidReportException("Could not get reference for report
id: " + reportId);
    }

    // Get the Calculated Fields
    //-----
    cfStatement = conn.prepareStatement(cfQuery);
    cfStatement.setInt(1, reportId);
    ResultSet cfResult = cfStatement.executeQuery();

```

Report.java

```

        while (cfResult.next()) {
            CalculatedField cf = new
CalculatedField(cfResult.getInt("CALCULATED_FIELD_ID"));

            cf.setReportId(cfResult.getInt("REPORT_ID"));
            cf.setDDName(cfResult.getString("NAME"));
            cf.setCalculation(cfResult.getString("CALCULATION"));
            cf.setOrderRank(cfResult.getInt("ORDER_RANK"));
            cf.setDisplayName(cf.getDDName(), cf.getCalculation(), conn);
            r.calculatedFields.add(cf);
        }

        // Get a reference of the objects that make a report
        //-----
        if (r.sortOrderId != 0) r.so = SortOrder.getReference(conn,
r.sortOrderId);
        r.fg = FilterGroup.getReference(conn, r.filterGroupId);
        if (r.layoutId != 0) r.l = Layout.getReference(conn,
String.valueOf(r.layoutId));
    }
    catch (Exception e) {
        Z.log.writeToLog(Log.ERROR, "Error in method getReference:" + e);
        throw e;
    }
    finally {
        //close statements
        if (statement != null) statement.close();
        if (cfStatement != null) cfStatement.close();
    }
    return r;
}

```

```

/**
 * getReference( Connection conn, String userId, String title, String
reportType, String outputType )
 * @param conn
 * @param title
 * @param reportType
 * @param outputType
 * @return Report
 * @throws InvalidReportException
 *
 * creates a static reference of Report for a given title
 */
public static Report getReference(Connection conn, String userId, String name,
String reportType, String outputType) throws InvalidReportException, Exception {
    Report r = null;

    // check if we have a connection
    if (conn == null) throw new Exception("Null connection");

    String query = "select          REPORT_ID, " +
        "          LAYOUT_ID, " +
        "          SORT_ORDER_ID, " +
        "          FILTER_GROUP_ID, " +
        "          OUTPUT_TYPE_NAME, " +
        "          REPORT_TYPE_NAME, " +
        "          REPORT_GROUP_ID, " +
        "          TITLE, " +
        "          DESCRIPTION, " +
        "          SECURITY_GROUP_ID, " +
        "          SECURITY_USER_ID, " +

```

```

                                Report.java
"    DATE_CREATED, " +
"    LAST_DATE_UPDATED, " +
"    LAST_UPDATED_BY_USER, " +
"    CREATED_BY_USER, " +
"    TITLE_MAP_KEY, " +
"    REPORT_LINK_URL " +
" from report " +
" where SECURITY_USER_ID = ? " +
" and    TITLE = ? " +
" and    REPORT_TYPE_NAME = ? " +
" and    OUTPUT_TYPE_NAME = ? " +
" order by REPORT_ID ";

String cfQuery = "select cf.CALCULATED_FIELD_ID, " +
" cf.REPORT_ID, " +
" cf.NAME, " +
" cf.CALCULATION " +
" from CALCULATED_FIELD cf, REPORT r " +
" where cf.report_id = r.report_id " +
" and r.SECURITY_USER_ID = ? " +
" and r.TITLE = ? " +
" and r.REPORT_TYPE_NAME = ? " +
" and r.OUTPUT_TYPE_NAME = ? " +
" order by cf.order_rank ";

PreparedStatement statement = null;
PreparedStatement cfStatement = null;

try {

    statement = conn.prepareStatement(query);
    int i = 1;

    statement.setString(i++, userId);
    statement.setString(i++, name);
    statement.setString(i++, reportType);
    statement.setString(i++, outputType);
    ResultSet result = statement.executeQuery();

    // Get the information associated with this report_id
    //-----
    if (result.next()) {
        int reportId = result.getInt("REPORT_ID");

        r = new Report(reportId);
        r.layoutId = result.getInt("LAYOUT_ID");
        r.sortOrderId = result.getInt("SORT_ORDER_ID");
        r.filterGroupId = result.getInt("FILTER_GROUP_ID");
        r.outputTypeName = result.getString("OUTPUT_TYPE_NAME");
        r.setReportType(result.getString("REPORT_TYPE_NAME"));
        r.reportGroupId = result.getInt("REPORT_GROUP_ID");
        r.name = result.getString("TITLE");
        r.title = result.getString("DESCRIPTION");
        r.titleMapKey = result.getLong("TITLE_MAP_KEY");
        r.securityUserId = result.getString("SECURITY_USER_ID");
        r.securityGroupId = result.getString("SECURITY_GROUP_ID");
        r.createdOn = result.getTimestamp("DATE_CREATED");
        r.updatedOn = result.getTimestamp("LAST_DATE_UPDATED");
        r.updatedBy = result.getString("LAST_UPDATED_BY_USER");
        r.createdBy = result.getString("CREATED_BY_USER");
        r.reportListURL = result.getString("REPORT_LINK_URL");
    }
    else {

```

```

Report.java
        throw new InvalidReportException("Could not get reference for
user/title/reportType/outputType " +
        userId + "/" + name + "/" + reportType + "/" + outputType);
    }

    // Get the Calculated Fields
    //-----
    cfStatement = conn.prepareStatement(cfQuery);
    i = 1;
    cfStatement.setString(i++, userId);
    cfStatement.setString(i++, name);
    cfStatement.setString(i++, reportType);
    cfStatement.setString(i++, outputType);
    ResultSet cfResult = cfStatement.executeQuery();

    while (cfResult.next()) {
        CalculatedField cf = new
CalculatedField(cfResult.getInt("CALCULATED_FIELD_ID"));

        cf.setReportId(cfResult.getInt("REPORT_ID"));
        cf.setDDName(cfResult.getString("NAME"));
        cf.setCalculation(cfResult.getString("CALCULATION"));
        cf.setOrderRank(cfResult.getInt("ORDER_RANK"));
        cf.setDisplayName(cf.getDDName(), cf.getCalculation(), conn);
        r.calculatedFields.add(cf);
    }

    // Get a reference of the objects that make a report
    //-----
    if (r.sortOrderId != 0) r.so = SortOrder.getReference(conn,
r.sortOrderId);
    r.fg = FilterGroup.getReference(conn, r.filterGroupId);
    if (r.layoutId != 0) r.l = Layout.getReference(conn,
String.valueOf(r.layoutId));
    }
    catch (Exception e) {
        throw e;
    }
    finally {
        //close statements
        if (statement != null) statement.close();
        if (cfStatement != null) cfStatement.close();
    }
    return r;
}

/**
 * public static void deleteRemovedDDE()
 * @param String ddeName
 * @param Connection conn
 *
 * deletes the dde from sort_order, sort_order_field, filter and filter_criteria
tables
 *
 */

public static void deleteRemovedDDE(String ddeName, Connection conn) throws
Exception {

    PreparedStatement fcStmt = null;
    PreparedStatement cfStmt = null;
    PreparedStatement fstmt = null;
    PreparedStatement sofStmt = null;

```



```

Report.java
PreparedStatement soStmt = null;

String fcQuery = "delete from filter_criteria where filter_id in " +
    " (select filter_id from filter where dd_name = ?) ";

String fQuery = "delete from filter where dd_name = ? ";

String cfQuery = "delete from calculated_field where name = ? ";

String sofQuery = "delete from sort_order_field where sort_order_id in " +
    " (select sort_order_id from sort_order where dd_name = ?)
";

try {
    fcStmt = conn.prepareStatement(fcQuery);
    fStmt = conn.prepareStatement(fQuery);
    sofStmt = conn.prepareStatement(sofQuery);
    cfStmt = conn.prepareStatement(cfQuery);
    cfStmt.setString(1, ddeName);
    fcStmt.setString(1, ddeName);
    fStmt.setString(1, ddeName);
    sofStmt.setString(1, ddeName);
    cfStmt.executeUpdate();
    fcStmt.executeUpdate();
    fStmt.executeUpdate();
    sofStmt.executeUpdate();
}
catch (Exception e) {
    Z.log.writeToLog(Z.log.ERROR, "One of these failed:");
    Z.log.writeToLog(Z.log.ERROR, fcQuery);
    Z.log.writeToLog(Z.log.ERROR, fQuery);
    Z.log.writeToLog(Z.log.ERROR, sofQuery);
    throw e;
}
finally {
    if (cfStmt != null)        fcStmt.close();
    if (fcStmt != null)        fcStmt.close();
    if (fStmt != null)         fStmt.close();
    if (sofStmt != null)       sofStmt.close();
    if (soStmt != null)        soStmt.close();
}
}

/**
 * executeTransaction( String user_id )
 * @param user_id
 * @throws java.lang.exception
 *
 * Transactional interface.  Executes any pending database changes.
 *
 * !!NOTE!! - Requires that a connection be set with a call to
 *             setConnection() prior to usage.
 */

//***** TO DO: Save calculated
fields!!!!!!!!!!!!!!!!!!!!!!

public void executeTransaction(String userId) throws Exception {

    String xSortOrderId = null;
    String xLayoutId = null;
    String query = null;
    PreparedStatement statement = null;

```

```

String bv = null;

String SYSDATE = Z.dbms.CURRENT_DATE();

// make sure there is a transaction to execute
if (!modifiedData && !newData && !deletedData) return;

// no database record exists, so do nothing
//-----
if (deletedData && newData) {
    newData = false;
    modifiedData = false;
    deletedData = false;
    return;
}

if (conn == null) throw new Exception("Null connection");

try {
    // Delete has priority (since the other flags would be irrelevant)
    //-----
    if (deletedData) {

        // ----- delete report record -----
        query = "delete from report where report_id = ? ";
        statement = conn.prepareStatement(query);
        statement.setInt(1, this.reportId);

        //get bindValues
        bv = PreparedStatementProxy.getBindValues(statement);
        if (bv != null) query = query + "<br>(" + bv + ")";

        Z.m.deleteTitle(conn, titleMapKey);
        statement.executeUpdate();
        SystemLog.logTransaction(conn,
            SystemLogType.DELETE_REPORT,
            query.toString());
        statement.close();

        this.fg.delete(conn, userId, fg.getReference(conn,
this.filterGroupId));
        if (this.l != null) this.l.delete(conn, userId,
String.valueOf(this.layoutId));
        if (this.so != null) this.so.delete(conn, userId, this.sortOrderId);

    } // end delete

    // Insert comes next
    //-----
    else if (newData) {

        Z.log.writeToLog(this, Log.DEBUG4, "About to call Insert Report");

        if (this.l != null) {

            //Z.probe(" THE LAYOUT IS NOT NULL !!!!!!!");
            //Z.probe(" THE LAYOUT ID is-----> !!!!!!!" +
this.l.getLayoutId());
            this.l.insert(conn, userId);
        }
        if (this.so != null) this.so.insert(conn, userId);
        if (this.fg != null) this.fg.insert(conn, userId);
    }
}

```

Report.java

```

query = "insert into report " +
"(" +
"    REPORT_ID, " + //1
"    LAYOUT_ID, " + //2
"    SORT_ORDER_ID, " + //3
"    FILTER_GROUP_ID, " + //4
"    OUTPUT_TYPE_NAME, " + //5
"    REPORT_TYPE_NAME, " + //6
"    REPORT_GROUP_ID, " + //7
"    TITLE, " + //8
"    DESCRIPTION, " + //9
"    SECURITY_GROUP_ID, " + //10
"    SECURITY_USER_ID, " + //11
"    DATE_CREATED, " + // SYSDATE
"    LAST_DATE_UPDATED, " + // SYSDATE
"    LAST_UPDATED_BY_USER, " + //12
"    CREATED_BY_USER, " + //13
"    REPORT_LINK_URL, " + // 14
"    TITLE_MAP_KEY " +
" )" +
" values ( ?,?,?,?,?," +
" ?,?,?,?,?," +
" ?, " + SYSDATE + ", " + SYSDATE + ",?," +
" ?,?,?,? ) ";

int newReportId = Integer.parseInt(this.getSequence());

// sort order is optional -- can be null
if (this.so != null) xSortOrderId =
string.valueOf(so.getLatestSortOrderId());
if (this.l != null) xLayoutId = this.l.getLayoutId();

statement = conn.prepareStatement(query);
int index = 1;
statement.setInt(index++, newReportId);
statement.setString(index++, xLayoutId);
statement.setString(index++, xSortOrderId);
statement.setInt(index++, this.fg.getLatestFilterGroupId());
statement.setString(index++, this.outputTypeName);
statement.setString(index++, this.reportTypeName);
statement.setInt(index++, this.reportGroupId);
statement.setString(index++, this.name);
statement.setString(index++, this.title);
statement.setString(index++, this.securityGroupId);
statement.setString(index++, this.securityUserId);
statement.setString(index++, userId);
statement.setString(index++, userId);
statement.setString(index++, this.reportListURL);
this.titleMapKey = Z.m.getNewTitleKey();
statement.setLong(index++, this.titleMapKey);

//get bindValues
bv = PreparedStatementProxy.getBindValues(statement);
if (bv != null) query = query + "<br>(" + bv + ")";

Z.m.insertTitle(conn, this.titleMapKey, this.locale, this.title,
userId);

statement.executeUpdate();
SystemLog.logTransaction(conn,
    SystemLogType.ADD_REPORT,
    query.toString());

```

```

Report.java
statement.close();
} // end insert

// update when necessary
//-----
else if (modifiedData) {
    if (this.so != null) {
        if (this.sortOrderId == 0) this.so.insert(conn, userId);
        else this.so.update(conn, userId, so.getReference(conn,
this.sortOrderId));
    }
    if (this.fg != null) {
        if (this.filterGroupId == 0) this.fg.insert(conn, userId);
        else this.fg.update(conn, userId, fg.getReference(conn,
this.filterGroupId));
    }
    if (this.l != null) {
        if (this.layoutId == 0) this.l.insert(conn, userId);
        else this.l.update(conn, userId, String.valueOf(this.layoutId));
    }
}

query = "update report set " +
"    LAYOUT_ID = ?," + //1
"    SORT_ORDER_ID = ?," + //2
"    FILTER_GROUP_ID = ?," + //3
"    OUTPUT_TYPE_NAME = ?," + //4
"    REPORT_TYPE_NAME = ?," + //5
"    REPORT_GROUP_ID = ?," + //6
"    TITLE = ?," + //7
"    DESCRIPTION = ?," + //8
"    SECURITY_GROUP_ID = ?," + //9
"    SECURITY_USER_ID = ?," + //10
"    LAST_DATE_UPDATED = " + SYSDATE + "," + // SYSDATE
"    LAST_UPDATED_BY_USER = ?," + //11
"    REPORT_LINK_URL = ?," + //12
"    TITLE_MAP_KEY = ? " +
"where report_id = ? "; //13

statement = conn.prepareStatement(query);
int index = 1;
statement.setInt(index++, this.layoutId);
statement.setInt(index++, this.sortOrderId);
statement.setInt(index++, this.filterGroupId);
statement.setString(index++, this.outputTypeName);
statement.setString(index++, this.reportTypeName);
statement.setInt(index++, this.reportGroupId);
statement.setString(index++, this.name);
statement.setString(index++, this.title);
statement.setString(index++, this.securityGroupId);
statement.setString(index++, this.securityUserId);
statement.setString(index++, userId);
statement.setString(index++, this.reportListURL);
statement.setInt(index++, this.reportId);
statement.setLong(index++, this.titleMapKey);

//get bindValues
bv = PreparedStatementProxy.getBindValues(statement);
if (bv != null) query = query + "<br>(" + bv + ")";

```

```

Report.java
Z.m.updateTitle(conn, this.titleMapKey, this.locale, this.title,
userId);
    statement.executeUpdate();
    SystemLog.logTransaction(conn,
        SystemLogType.UPDATE_REPORT,
        query.toString());
    statement.close();

    } //end update

}
catch (Exception e) {
    Z.log.writeToLog(Log.ERROR, "Error in method executeTransaction:" + e);
    ErrorWriter.write(e, ErrorWriter.LOG);
    throw e;
}
finally {
    if (statement != null) statement.close();

    // clear transaction buffers
    newData = false;
    modifiedData = false;
    deletedData = false;
}
}

/**
 * getNextResult()
 * @return HashMap
 * * the returned hashmap of results is one record, keyed by DDNAME
 */
public HashMap getNextResult(SesameSession session) throws Exception {

    String key = "";
    String elementName = "";
    String elementType = "";
    DDentry thisDDE = null;
    ArrayList leList = null; // list of layout elements
    HashMap row = new HashMap();
    int counter = 0;

    try {
        if (resultEnum == null)
            throw new Exception("The there were no details returned for the
following block of keys :"+
                                + this.blockKeys);

        // The enumeration is out of elements so get the next block of page
results
//-----
        while (resultEnum != null
            && !resultEnum.hasMoreElements()
            && (this.elementStartAt + 1) < this.resultCount) {

            resultBlock.clear();
            if (this.resultCount == (this.elementStartAt + 1)) {
                this.pageKeys.clear();
                break;
            }
            getNextInternalBlock(session);

```

Report.java

```

    }

    // Get the row from the enumeration
    //-----
    if (resultEnum != null && resultEnum.hasMoreElements()) {
        row = (HashMap) resultBlock.get(resultEnum.nextElement());
        this.elementStartAt++;
    } else {
        Z.log.writeToLog(Log.DEBUG, this, "getNextResult is returning a
null...");
    }

    }
    catch (Exception e) {
        Z.log.writeToLog(Log.ERROR, this, "Error in method getNextResult:" +
e);
        ErrorWriter.write(e, ErrorWriter.LOG);
        throw e;
    }
    finally {
        return row;
    }
}

/**
 * getNextInternalBlock()
 * @param SesameSession session
 */
private void getNextInternalBlock(SesameSession session) throws Exception {

    int highCnt = 0;
    int retrieveCount = 0;

    resultBlock = new ValidationList();
    this.blockKeys.clear();

    try {

        // Figure out how many records to iterate in the block
        //-----

        this.startedPageBlockCalc = true; // just set the starting one once
until next internalblock

        // Deal with unlimited results - process the next unlimitedResultBlock
        if (this.unlimitedResults) {

            Persist persist = (Persist) session.getAttribute(session.getId() +
"_REPORT_IDS");

            int readThisMany = 0;

            int currentTopLimit = this.unlimitedBlock * MAX_PERSIST_LIST_SIZE;

            // Z.probe("@@@@@@ this.elementStartAt = " + this.elementStartAt );
            // Z.probe("@@@@@@ this.unlimitedBlock*MAX_PERSIST_LIST_SIZE = " +
this.unlimitedBlock * " * " + MAX_PERSIST_LIST_SIZE + " = " + currentTopLimit);
            // Z.probe("@@@@@@ this.resultCount = " + this.resultCount );
            if (this.elementStartAt < (currentTopLimit)
                && this.elementStartAt < this.resultCount) {

                this.unlimitedBlock++;
            }
        }
    }
}

```



```

Report.java
// Here we read from persist startAt to the smallest of:
//     a) currentTopLimit
//     b) this.maxRows
//     c) this.resultCount
//
// For Example:
//     Suppose the maximum that can be read from persist is
//     MAX_PERSIST_LIST_SIZE = 5000
//     this.maxRows (set from AppDefault) = 7500
//     and the result count is 8000
//
//     We will read from 0 to 5000 ids from persist the first
iteration
//     Then from 5000 to 7,500 (maxRows)
//
//     If maxrows is set to 10,000 then:
iteration
//     We will read from 0 to 5000 ids from persist the first
//     Then from 5000 to 8000 (resultCount)
this.resultCount) { if (this.elementStartAt + MAX_PERSIST_LIST_SIZE >
    if (this.checkMaxRows) {
        if (this.maxRows > this.resultCount)
            readThisMany = this.resultCount;
        else
            readThisMany = this.maxRows;
    } else {
        readThisMany = this.resultCount;
    }
} else {
    if (this.checkMaxRows) {
        if (this.elementStartAt + MAX_PERSIST_LIST_SIZE >
this.maxRows)
            readThisMany = this.maxRows;
        else
            readThisMany = MAX_PERSIST_LIST_SIZE;
    } else {
        readThisMany = MAX_PERSIST_LIST_SIZE;
    }
}

    this.internalBlockStartElement = 0;
    // Z.probe("RPT:@@@@@@@@@@
pageKeys.size=persist.read("+elementStartAt+", "+readThisMany+") :");
    this.pageKeys = persist.read(elementStartAt, readThisMany);
}
}
// Z.probe("RPT:@@@@@@@@@@@ pageKeys.size():"+pageKeys.size());
// Z.probe("RPT:@@@@@@@@@@@ pageKeys:"+pageKeys);
// Z.probe("RPT:@@@@@@@@@@@
internalBlockStartElement:"+internalBlockStartElement);
// Z.probe("RPT:@@@@@@@@@@@ INTERNAL_BLOCK_SIZE:"+INTERNAL_BLOCK_SIZE);

    if (this.pageKeys.size() - (internalBlockStartElement + 1) >
INTERNAL_BLOCK_SIZE)
        highCnt = INTERNAL_BLOCK_SIZE + internalBlockStartElement;
    else
        highCnt = pageKeys.size();

    // Z.probe("@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@");

```

```

Report.java
// Z.probe("RPT:@@@@@@@@@
internalBlockStartElement:"+internalBlockStartElement);
// Z.probe("RPT:@@@@@@@@@ highCnt:"+highCnt);

// Create the list to be used for that block
//-----

for (int k = internalBlockStartElement; k < highCnt; k++) {
    this.blockKeys.add((String) pageKeys.get(k));
    this.internalBlockStartElement++;
}

//Z.probe("RPT:@@@@@@@@@ pageKeys.size():"+pageKeys.size());
//Z.probe("RPT:@@@@@@@@@ blockKeys:"+blockKeys);
//Z.probe("RPT:@@@@@@@@@ blockKeys.size():"+blockKeys.size());

resultBlock = getSelect(conn, this.blockKeys, session);

if (resultBlock.isEmpty()) resultEnum = null;
else resultEnum = resultBlock.keys();

//Z.probe("RPT:@@@@@@@@@ resultEnum:"+resultEnum );
}
catch (Exception e) {
    Z.log.writeToLog(Log.ERROR, this, "Error in method getNextBLOCKResult:"
+ e);
    ErrorWriter.write(e, ErrorWriter.LOG);
    throw e;
}

}

public int getResults(Connection conn,
                    boolean isNewPersist,
                    int startAt,
                    int resultsOnPage,
                    SesameSession session) throws Exception {

    ReportPersist rPersist = null;
    Object temp = null;
    String handle = null;

    Z.probe("In Report.getResults() ");
    try {

        handle = session.getId() + "_REPORT_IDS";
        if ((temp = session.getAttribute(handle)) == null) {
            rPersist = new ReportPersist(session);
            session.setAttribute(handle, rPersist);
            this.isNewPersist = true;
        } else rPersist = (ReportPersist) temp;

        return doGetResults(conn, isNewPersist, startAt, resultsOnPage, session,
rPersist, false);
    }
    catch (Exception e) {
        throw e;
    }
}

public int getAttachAlert(Connection conn,
                        boolean isNewPersist,
                        int startAt,
                        int resultsOnPage,

```

Report.java
SesameSession session) throws Exception {

```

ReportPersist rPersist = null;
Object temp = null;
String handle = null;

Z.probe("In Report.getAttachAlert() ");
try {
    handle = session.getId() + "_REPORT_IDS";
    if ((temp = session.getAttribute(handle)) == null) {
        rPersist = new ReportPersist(session);
        session.setAttribute(handle, rPersist);
        this.isNewPersist = true;
    } else rPersist = (ReportPersist) temp;

    return doGetResults(conn, isNewPersist, startAt, resultsOnPage, session,
rPersist, true);
}
catch (Exception e) {
    throw e;
}
}

public int getResults(Connection conn,
                    boolean isNewPersist,
                    int startAt,
                    int resultsOnPage,
                    SesameSession session,
                    String cliPersistHandle) throws Exception {
    CLIPersist cPersist = null;

    try {
        if (TextManager.isStringVisible(cliPersistHandle)) {
            cPersist = new CLIPersist(cliPersistHandle);
            if (resultsOnPage == -1){
                session.setAttribute("REPORT_IDS", cPersist);
                this.isNewPersist = true;
            }
        }

        return doGetResults(conn, isNewPersist, startAt, resultsOnPage, session,
cPersist, false);
    }
    catch (Exception e) {
        throw e;
    }
}

/**
 * doGetResults()
 * @param conn
 * @param isNewPersist
 * @param startAt
 * @param resultsPerPage
 * @param userId
 *
 * gets the results into the class object rs (a result set)
 */
private int doGetResults(Connection conn,
                        boolean isNewPersist,
                        int startAt,

```

```

                                Report.java
                                int resultsOnPage,
                                SesameSession session,
                                Persist persist,
                                boolean doAttachAlert) throws Exception {

//Z.probe("In Report.doGetResults() ");
boolean doCLI = false;
int lastPageBlock = 0;
int retrieveCount = 0;
int iterations = 0;
int highCnt = -1;
Object temp = null;
ArrayList keys = null;

// Get the list from the DB or Persist ---- save block on n pages in
persist
//-----
try {
    this.isNewPersist = isNewPersist;

    this.setConnection(conn);
    if (startAt < 1) startAt = 1;
    this.elementStartAt = startAt - 1;
    this.setResultPerPage(resultsOnPage);

    // Determine the pageBlock -- only read list for that pageBlock
    retrieveCount = this.PAGES_PER_PAGEBLOCK * this.resultPerPage;

    // New page
    if (pageKeys != null) this.pageKeys.clear();
    else this.pageKeys = new ArrayList();

    if (this.isNewPersist) {
        // WE GET 10 pages of keys ONLY...
        //Z.probe("$$$$ NEW PERSIST  $$$$");
        this.pageBlock = 1;
        this.getList(conn, session, persist, doAttachAlert);
    }
    else {
        //Z.probe("$$$$ EXISTING PERSIST  $$$$");
        //Z.probe("this.PAGES_PER_PAGEBLOCK"+this.PAGES_PER_PAGEBLOCK);
        //Z.probe("this.resultPerPage"+this.resultPerPage);
        //Z.probe("THIS.ELEMENT_STARTAT = " + this.elementStartAt);
        //Z.probe("retrieveCount = " +retrieveCount);
        //Z.probe("this.pageBlock = " +this.pageBlock );

        double calcPageCnt = ((double) (this.elementStartAt + 1) / (double)
retrieveCount);

        //Z.probe("calcPageCnt " + calcPageCnt);

        try {
            if (calcPageCnt > this.pageBlock) {
                this.pageBlock++;
                getList(conn, session, persist, doAttachAlert); // the next
pageBlock of keys
            }
        } // no more keys for the current page -- go out and get more keys
        catch (IOException ioe) {
            this.pageBlock++;

```

```

                                Report.java
                                this.getList(conn, session, persist, doAttachAlert);
                                //Z.probe("RPT:!!!!!!!!!!!!KEYS FOR NEXT PAGEBLOCK:"+keys);
                                }
                                }

                                //Z.probe("RPT:!!!!!!!!!!!!persist.size():"+ persist.getDataSize() );
                                //Z.probe("RPT:!!!!!!!!!!!!this.resultCount:"+this.resultCount);
                                //Z.probe("RPT:!!!!!!!!!!!!startAt:"+startAt);
                                //Z.probe("RPT:!!!!!!!!!!!!elementStartAt:"+this.elementStartAt);
                                //Z.probe("RPT:!!!!!!!!!!!!INTERNAL_BLOCK_SIZE:"+INTERNAL_BLOCK_SIZE);

                                if (persist == null) return 0;
                                int size = persist.getDataSize();

                                if (size == 0) return 0;

                                // Figure out how many records to iterate on the screen - from startAT
to highcnt
                                //-----
                                -
                                this.internalBlockStartElement = 0;
                                this.unlimitedBlock = 0;

                                if (unlimitedResults) {
                                    highCnt = size;
                                    this.unlimitedBlock = 1;
                                }
                                else {
elementStartAt)
                                highCnt = ((size - elementStartAt) < this.resultPerPage) ? (size -
                                                                :
                                this.resultPerPage;
                                this.pageKeys = persist.read(elementStartAt, elementStartAt +
highCnt);
                                }

                                //Z.probe("-----");
                                //Z.probe("RPT:!!!!!!!!!!!! highCnt:" + highCnt);

                                //Z.probe("RPT:!!!!!!!!!!!!this.internalBlockStartElement:"+this.internalBlockStartEle
ment);

                                this.startedPageBlockCalc = false;
                                if (!doAttachAlert) getNextInternalBlock(session);
                                }
                                catch (Exception e) {
                                    Z.log.writeToLog(Log.ERROR, this, "Error in method getResults:" + e);
                                    ErrorWriter.write(e, ErrorWriter.LOG);
                                    throw e;
                                }
                                finally {
                                    if (doAttachAlert) return this.totalAttachmentSize;
                                    else return this.resultCount;
                                }
                                }

                                /**
                                * getCompleteIdList()
                                * @param Connection conn
                                * @param int startAt
                                * @param SesameSession session
                                *

```

```

    * returns the complete ID list
    **/
    public RelationshipGroupPersist getCompleteIdList(Connection conn,
                                                    SesameSession session) throws
Exception {
    int lastPageBlock = 0;
    int retrieveCount = 0;
    int iterations = 0;
    int highCnt = -1;
    int startAt = 1;
    ArrayList keys = null;
    Object temp = null;
    RelationshipGroupPersist rgPersist = null;

    try {
        this.isNewPersist = true;
        if ((temp = session.getAttribute("p_prob_array")) == null) {
            rgPersist = new RelationshipGroupPersist(session);
            session.setAttribute("p_prob_array", rgPersist);
        }
        else rgPersist = (RelationshipGroupPersist) temp;

        // Get the list from the DB or Persist ---- save full list in persist
        //-----
        this.setConnection(conn);
        this.elementStartAt = startAt - 1;
        this.setResultPerPage(-1);
        //Z.probe("!!!!!!!!!!!!!!!!!!!!GETTING RELATIONSHIP GROUP COMPLETE ID
LIST!!!!!!!!!!!!!!!!!!!!!!!!!!!!");

        this.getList(conn, session, rgPersist, false);

    }
    catch (Exception e) {
        Z.log.writeToLog(Log.ERROR, this, "Error in method getResults:" + e);
        throw e;
    }
    finally {
        session.setAttribute("p_prob_array", rgPersist);
        return rgPersist;
    }
}

/**
 * getList()
 * @param Connection conn
 * @param SesameSession session
 * @return ArrayList
 */
public ArrayList getList(Connection conn, SesameSession session, Persist
persist, boolean doAttachPersist) throws Exception {
    try {
        this.cleanFlags();
        return getMasterList(conn, session, persist, doAttachPersist);
    }
    catch (Exception e) {
        Z.log.writeToLog(Log.ERROR, this, "Error in method getList:" + e);
        throw e;
    }
}

/**

```


Report.java

```

* getSummaryReport()
* @param Connection conn
* @param SesameSession session
* @return ArrayList
*/
public ArrayList getSummaryReport(Connection conn, SesameSession session) throws
Exception {
    try {
        this.emptyStringSpacer = "-";
        this.cleanFlags();
        this.setSummaryReport(true);
        return getMasterList(conn, session, null, false);
    }
    catch (Exception e) {
        Z.log.writeToLog(Log.ERROR, this, "Error in method getList:" + e);
        throw e;
    }
}

/**
* getStatusOnDate()
* @param Connection conn
* @param SesameSession session
* @return ArrayList
*/
public ArrayList getStatusOnDate(Connection conn, SesameSession session) throws
Exception {
    try {
        this.emptyStringSpacer = "-";
        this.cleanFlags();
        this.setSummaryReport(true);
        this.setStatusOnDate(true);
        return getMasterList(conn, session, null, false);
    }
    catch (Exception e) {
        Z.log.writeToLog(Log.ERROR, this, "Error in method getList:" + e);
        throw e;
    }
}

/**
* getProductOnDate()
* @param Connection conn
* @param SesameSession session
* @return ArrayList
*/
public ArrayList getProductOnDate(Connection conn, SesameSession session) throws
Exception {
    try {
        this.emptyStringSpacer = "-";
        this.cleanFlags();
        this.setSummaryReport(true);
        this.setProductOnDate(true);
        return getMasterList(conn, session, null, false);
    }
    catch (Exception e) {
        Z.log.writeToLog(Log.ERROR, this, "Error in method getList:" + e);
        throw e;
    }
}

/**
* getReleaseOnDate()

```

Report.java

```

* @param Connection conn
* @param SesameSession session
* @return ArrayList
*/
public ArrayList getReleaseOnDate(Connection conn, SesameSession session) throws
Exception {
    try {
        this.emptyStringSpacer = "-";
        this.cleanFlags();
        this.setSummaryReport(true);
        this.setReleaseOnDate(true);
        return getMasterList(conn, session, null, false);
    }
    catch (Exception e) {
        Z.log.writeToLog(Log.ERROR, this, "Error in method getList:" + e);
        throw e;
    }
}

/**
* getStatusByDate()
* @param Connection conn
* @param SesameSession session
* @return ArrayList
*/
public ArrayList getStatusByDate(Connection conn, SesameSession session, int
timeInterval) throws Exception {
    try {
        this.emptyStringSpacer = "-";
        this.cleanFlags();
        this.setSummaryReport(true);
        this.setStatusByDate(true);
        this.setTimeInterval(timeInterval);
        return getMasterList(conn, session, null, false);
    }
    catch (Exception e) {
        Z.log.writeToLog(Log.ERROR, this, "Error in method getList:" + e);
        throw e;
    }
}

/**
* getProductByDate()
* @param Connection conn
* @param SesameSession session
* @return ArrayList
*/
public ArrayList getProductByDate(Connection conn, SesameSession session) throws
Exception {
    try {
        this.emptyStringSpacer = "-";
        this.cleanFlags();
        this.setSummaryReport(true);
        this.setProductByDate(true);
        return getMasterList(conn, session, null, false);
    }
    catch (Exception e) {
        Z.log.writeToLog(Log.ERROR, this, "Error in method getList:" + e);
        throw e;
    }
}

/**

```

Report.java

```

* getReleaseByDate()
* @param Connection conn
* @param SesameSession session
* @return ArrayList
*/
public ArrayList getReleaseByDate(Connection conn, SesameSession session) throws
Exception {
    try {
        this.emptyStringSpacer = "-";
        this.cleanFlags();
        this.setSummaryReport(true);
        this.setReleaseByDate(true);
        return getMasterList(conn, session, null, false);
    } catch (Exception e) {
        Z.log.writeToLog(Log.ERROR, this, "Error in method getList:" + e);
        throw e;
    }
}

```

```

/*****
**

```

```

****
*

```

```

* getMasterList()
* @return ArrayList of ids/ hashmaps
*
* Gets list of IDs for a detailed/ custom / quicklist report
* OR
* Does summary report and returns a HashMap of tag/value pairs
*

```

```

****
**

```

```

****
**/

```

```

private ArrayList getMasterList(Connection conn,
                                SesameSession session,
                                Persist persist,
                                boolean doAttachAlert) throws Exception {

```

```

// Used for processing the query string
//-----
boolean isTimeSeries = false;
boolean isProductOrReleaseChart = false;
boolean useNotExists = false;
boolean appendChartData = true;
boolean wildcards = false;
boolean hasNullCriteria = false;
boolean skipComma = false;
boolean doCalcFields = false;
boolean gotDate = false;
boolean addReleasesSql = false;

```

```

int i = 0;
int criteriaSize = 0;
int fromCnt = 0; // count of from tables
int paramCnt = 0; // parameter count
int udfCnt = 0; // udf count
int sofCnt = 0; // sort order field count
int joinCnt = 0; // count of joins made

```

```

Report.java
int tableAliasCnt = 0; // count of the table aliases
int retrieveCount = 0;
String idCountAlias = "ID_COUNT";
String valueSufx = "_VALUE";
String alias = "";
String sortBy = "";
String thisTableName = "";
String rawValue = "";
String processedValue = "";
String temp = "";
String key = ""; // this is the ddName
String keywordHint = Z.dbms.indexHint("ITEM", "IX_ITEM1");

StringBuffer select = new StringBuffer();
StringBuffer gSelect = new StringBuffer(); // for the group by/ summary
reports
StringBuffer from = new StringBuffer(" from ");
StringBuffer where = new StringBuffer(" where ");
StringBuffer groupBy = new StringBuffer(" group by ");
String query = "";
String countQuery = "";
String tableAlias = ""; // table alias used in sorting
String luTableAlias = ""; // LookUp table alias used in sorting
String tableListAlias = ""; // table alias used in sorting UDF values for
the list table
String tableUdfAlias = "";
StringBuffer andUDF = new StringBuffer();
StringBuffer andKeyword = new StringBuffer();
StringBuffer andProblem = new StringBuffer();
StringBuffer andOther = new StringBuffer();
StringBuffer andSortOrder = new StringBuffer();
StringBuffer andItemType = new StringBuffer();
StringBuffer orderBy = new StringBuffer();
StringBuffer andExists = new StringBuffer();
StringBuffer notExists = new StringBuffer();
StringBuffer preCondition = new StringBuffer();
StringBuffer criteriaTableColumn = new StringBuffer();

Object tempObject = null;

// Other objects used in processing
//-----
ArrayList rawfc = new ArrayList(); // raw filter criteria (no wildcard
substition)
ArrayList fc = new ArrayList(); // filter criteria
ArrayList params = new ArrayList(); // the params
ArrayList paramsUDF = new ArrayList(); // the UDF params
ArrayList paramsKeyword = new ArrayList(); // the Keyword params
ArrayList paramsItemType = new ArrayList();
ArrayList paramsProblem = new ArrayList(); // the Problem Table params
ArrayList paramsOther = new ArrayList(); // the for storing Table params
for child tables used in the exist clause
ArrayList paramsExists = new ArrayList(); // the child Table params
ArrayList paramsPreCondition = new ArrayList(); // the preCondition params
ArrayList typesExists = new ArrayList(); // the types of
params
ArrayList paramsSortOrder = new ArrayList(); // the params for used to
match the UDF names when sorting on a UDF
ArrayList types = new ArrayList(); // the types
ArrayList typesUDF = new ArrayList(); // the UDF types
ArrayList typesPreCondition = new ArrayList(); // the preCondition types
ArrayList typesKeyword = new ArrayList(); // the Keyword types
ArrayList typesProblem = new ArrayList(); // the Problem Table types

```

```

Report.java
ArrayList typesItemType = new ArrayList();
ArrayList typesOther = new ArrayList(); // the other Table types
ArrayList typesSortOrder = new ArrayList(); // the types for used to match
the UDF names when sorting on a UDF

ArrayList results = new ArrayList(); // the resulting list of IDs in the
result set

SortOrderInfo sortOrderInfo = null;

ArrayList summaryVals = new ArrayList();

TreeMap soFields = new TreeMap(); // the sorted list of fields ---
the key is the sort order as a string
HashMap udfs = new HashMap(); // the referenced HashMap of UDFs
--- this should a reference from a serialized object rather than Database
HashMap distinctTables = new HashMap(); // used to track distinct tables for
the "from" & "where" clauses -- key is tableName
HashMap distinctLookupTables = new HashMap(); // used to track distinct
lookup tables for the "from" & "where" clauses -- key is tableName+lookupTableName
HashMap filterFields = null; // from data_dictionary where
filterCriteria = 'Y'
HashMap sortFields = null; // from data_dictionary where sort
order = 'Y'
HashMap chartableFields = null;
HashMap notExistClauses = new HashMap();
HashMap existClauses = new HashMap();
HashMap existParams = new HashMap();
HashMap existTypes = new HashMap();
HashMap existAndOrs = new HashMap();

ArrayList nullCriteriaList = new ArrayList();
HashMap problemColumns = new HashMap();
HashMap groupByAliases = null;
HashMap rows = null;
Timestamp criteriaTimestamp = null;
PreparedStatement statement = null;
PreparedStatement countStatement = null;
PreparedStatement sumAttachStatement = null;

Locale loc = session.getLocale();

DDEntry thisDDE = null; // a Data Dictionary Entry
DbTime dbt = null;
DbTime dbt2 = null;
DbTime dbtForTimeSeries = null;

SecurityUser su = null;

Filter f = new Filter(); // filter

JoinData jd = new JoinData();
ReportElement re = null;

try {

    if (this.isSummaryReport()) groupByAliases = new HashMap();
    else retrieveCount = this.PAGES_PER_PAGEBLOCK * this.resultPerPage;

    // set values of objects to be used
    //-----
    su = (SecurityUser) session.getAttribute("USER");

```

```

                                Report.java
chartableFields = Z.dictionary.getChartableFields();
filterFields = Z.dictionary.getFilterFields();
filterFields.putAll(chartableFields);
sortFields = Z.dictionary.getSortableFields();
sortFields.putAll(chartableFields);
udfs = Udf.getUdfMap(conn);

String hashCode = ""; // used to track the exists/ not exists clauses

HashMap orExists = new HashMap(); // to do an or exists for multi
criteria when one is a not exists

// LOOP THROUGH THE FILTERS TO FIGURE OUT THE BETWEEN DATES
//-----
HashMap origFilters = null;
if (this.fg != null) origFilters = this.fg.getFilters();
else origFilters = new HashMap();

Z.log.writeToLog(Z.log.DEBUG5,"THE ORIG FILTERS ARE
this.fg.getFilters(): " + origFilters);

Iterator ofi = origFilters.keySet().iterator();
HashMap betweenDates = new HashMap();
HashMap startDates = new HashMap();
HashMap filters = new HashMap();
String tempDDName = "";
Filter tmpf = null;
boolean removeFirstDate = false;
String stopTimeSeriesDate = "";
String startTimeSeriesDate = "";

this.suppressedMultiReleaseFilters.clear();

if (this.isProductByDate() || this.isProductOnDate()
    || this.isReleaseByDate() || this.isReleaseOnDate()){
    isProductOrReleaseChart = true;
}

if (this.isStatusByDate() || this.isProductByDate() ||
this.isReleaseByDate())
    isTimeSeries = true;

if (isTimeSeries || this.isStatusOnDate() || this.isProductOnDate() ||
this.isReleaseOnDate())
    isChart = true;

// Need to format data for a summary report -
// get initial DbTime object to figure db and user offset from GMT
//-----
dbt = new DbTime(session, conn);
dbt2 = new DbTime(session, conn);

if (isTimeSeries) dbtForTimeSeries = new DbTime(session,conn);

String baseName = "";

StringBuffer prSql = new StringBuffer("( select  i.item_id, ig.item2_id
"); /// the subQuery in the from clause -- for releases
ArrayList releaseCols = new ArrayList();

// for local processing

```



```

String tmpTable = "";
String tmpItemId = "";
String tmpColumn = "";
String tmpName = "";
String tmpDisplayType = "";
String tmpType = "";

while (ofi.hasNext()) {
    key = (String) ofi.next();
    if (key.indexOf("RANGE_START") > -1)
        baseName = key.substring(12);
    else if (key.indexOf("RANGE_STOP") > -1)
        baseName = key.substring(11);
    else
        baseName = key;

    // Only process valid filter fields
    //-----
    if ((thisDDE = (DDEEntry) filterFields.get(baseName)) == null)
continue;

    thisDDE.setName(key);
    f = this.fg.getFilter(key);

    //Z.probe("Checking the origFilters for key = " + key);

    tmpTable = thisDDE.getTableName() == null ? "" :
thisDDE.getTableName();

    if (TextManager.isStringVisible(thisDDE.getItemId()))
        tmpItemId = thisDDE.getItemId();

    if ("PROBLEM_RELEASE".equalsIgnoreCase(tmpTable) ||
"1".equalsIgnoreCase(tmpItemId)){
        tmpColumn = thisDDE.getColumnName();
        prFilters.put(tmpName, "");
        if (!addReleaseSql){
            addReleaseSql = true;
            prSql.append(", i."+tmpColumn + " ");
        }
    }
    if (this.hr) tmpTable = getHistoryEquivalent(tmpTable);

    tmpName = thisDDE.getName();
    tmpDisplayType = thisDDE.getDisplayType();

    // Track the multiple value release fields that need to be
suppressed
    // in a hashmap if DDName/Filter (this is originally for redback)
    //-----
    if (this.filterChildValue){
        if ("PROBLEM_RELEASE".equalsIgnoreCase(tmpTable)){
            this.suppressedMultiReleaseFilters.put(tmpName, f);
        }
    }

    //Z.probe("Checking ddeName = " + ddeName + " and it is a
searchField");

    rawfc = f.getFilterCriteriaResolved();
    criteriaSize = rawfc.size();

```

Report.java

```

// debug check...
if (criteriaSize == 0) {
    ArrayList fcTest = f.getFilterCriteria();
    if (fcTest.size() > 0) Z.log.writeToLog(Z.log.ERROR, "RPT:
resolved mismatch error:" + fcTest);
}

if (TextManager.isStringVisible(thisDDE.getColumnNames()))
    tmpColumn = TextManager.replace(thisDDE.getColumnNames(), "||",
Z.dbms.cat());

// for finding the optimal index
if (driverTable.equalsIgnoreCase(tmpTable)){

    // Do not process summary fields on charting queries -- they are
special.
    // do not use this index for STOP_UPDATE on piechart queries (a
<= not a between)

//-----
    // skip the hints for these chart queries --- they really slow
things down
//-----
    if (isChart){
        if (isTimeSeries){
            if ("START_UPDATE".equalsIgnoreCase(tmpName)){
                startTimeSeriesDate = (String)rawfc.get(0);
                continue; // skip this one
            }
            else if("STOP_UPDATE".equalsIgnoreCase(tmpName)){
                stopTimeSeriesDate = (String)rawfc.get(0);
            }
        }
        /* if ("STATUS_HIST".equalsIgnoreCase(ddeName)
        || "STATUS".equalsIgnoreCase(ddeName))
        continue;
        else if ("PRODUCT_NAME_HIST".equalsIgnoreCase(ddeName)
        || "PRODUCT_NAME".equalsIgnoreCase(ddeName))
        continue;
        else if ("RELEASE_FOUND_HIST".equalsIgnoreCase(ddeName)
        || "RELEASE_FOUND".equalsIgnoreCase(ddeName))
        continue;
        // else if (!("STOP_UPDATE".equalsIgnoreCase(ddeName) ||
"START_UPDATE".equalsIgnoreCase(ddeName)) )
        //     problemColumns.put(ddeColumn, f);
        */
    }
    else problemColumns.put(tmpColumn, f);
}

// if its not a date
if (!"DATE".equalsIgnoreCase(tmpDisplayType)) {

    // don't do the keyword criteria if its an attachment alert
    if ("KEYWORD".equalsIgnoreCase(tmpName) && doAttachAlert)
continue;

    else filters.put(tmpName, f);
    continue;
}

```

Report.java

```

// if its a date equivalence then add it
if ( !(tmpName.indexOf("RANGE_START") > -1
    || tmpName.indexOf("_SINCE") > -1
    || tmpName.indexOf("RANGE_STOP") > -1) ){
    filters.put(tmpName, f);
    continue;
}

// check for the same date column multiple times
if ((tmpf = (Filter) startDates.get(baseName)) == null) {
    startDates.put(baseName, f);
    filters.put(tmpName, f);
}

// already have one date so another means we use between (instead of
// clear out the last filter entry for this field and make the new
// filter entry a between with 2 dates -- the start date is [0],
// stop date is [1]
//-----
else {
    tempDDName = tmpf.getDDName();
    fc = new ArrayList();
    //Z.probe("BETWEEN DATE tmpDDNAME = " +
    //    tempDDName + " filter criteria = " +
    tmpf.getFilterCriteria());
    //Z.probe("BETWEEN DATE tmpDDNAME = " +
    //    tempDDName + " filter criteria(0) = " + (String)
    tmpf.getFilterCriteria().get(0));
    // set the criteria from the first found instance of this field
    // then set the criteria from the current -- make sure they are
    paired up

    removeFirstDate = false;

    if (tempDDName.indexOf("RANGE_START") > -1
        || tempDDName.indexOf("_SINCE") > -1) {
        fc.add(tmpf.getFilterCriteriaResolved().get(0));
        if (tmpName.indexOf("RANGE_STOP") > -1) {
            fc.add(rawfc.get(0));
            removeFirstDate = true;
        }
    }
    else if (tempDDName.indexOf("RANGE_STOP") > -1) {
        if (tmpName.indexOf("RANGE_START") > -1
            || tmpName.indexOf("_SINCE") > -1) {
            fc.add(rawfc.get(0));
            removeFirstDate = true;
        }
    }
    fc.add(tmpf.getFilterCriteriaResolved().get(0));
}

// set the criteria from this current instance of this field
// note that if the user laid out the screen properly there
// should be one START and one STOP only - so i will check to
enforce
// NOW if any DATE type has 2 FILTER CRITERIA and the ddName

```

contains the string

// STOP / START / SINCE it will be treated as a BETWEEN

```

//-----
if (removeFirstDate) {
    filters.remove(tempDDName); // remove the last
    f.setFilterCriteria(fc);
    filters.put(tmpName, f); // add the current
}
else {
    filters.put(tmpName, f);
    continue;
}
}
Z.log.writeToLog(Z.log.DEBUG5, "THE ORIG FILTERS ARE : "
+origFilters.toString());

Z.log.writeToLog(Z.log.DEBUG5, "THE PRE PROCESSED FILTERS ARE : "
+filters.toString());

OICCollection oil =
Z.dbms.getProblemTableOptimizedIndexHint(problemColumns, conn);

///-----DEBUG
STUFF-----
//Z.probe(" INDEX COLUMN ANALYSIS : problemColumns=" + problemColumns);
//if (oil != null) {
//Z.probe(" INDEX COLUMN ANALYSIS : oil.size()" + oil.size());
//if (oil.size() > 0) {
//for (int k = 0; k < oil.getIndexHintArray().length; k++)
//Z.probe(" INDEX COLUMN ANALYSIS : oil.getIndexHintArray()" +
oil.getIndexHintArray()[k]);
//
//for (int k = 0; k < oil.getTableAliasArray().length; k++)
//Z.probe(" INDEX COLUMN ANALYSIS : oil.getTableAliasArray()" +
oil.getTableAliasArray()[k]);
//}
//}
//-----DEBUG
STUFF-----
//!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

// variable params for history reporting
//-----
// if (this.hr) {
//     this.driverTable = "ITEM_HIST";
//     ITEM = "ITEM_HIST";
//     ITEM_TEXT = "ITEM_TEXT_HIST";
//     ITEM_UDF = "ITEM_UDF_HIST";

//Z.probe("~!@ - this.hr="+this.hr);
//Z.probe("~!@ - this.driverTable="+this.driverTable);
//Z.probe("~!@ - ITEM="+ITEM);
//Z.probe("~!@ - ITEM_TEXT="+ITEM_TEXT);
//Z.probe("~!@ - ITEM_UDF="+ITEM_UDF);
//} else
this.driverTable = "ITEM";

//*****

```

Report.java

```
//
// GET THE PRE-SORT INFO
//
SortOrderField sof = null;
SortOrder thisSo = null;
thisSo = this.so;

if ( ( (thisSo == null)
      || (thisSo.getFields().get("ID") == null) ) &&
!isSummaryReport() ) {

    sof = new SortOrderField();
    sof.setDDName("ID");

    if (thisSo != null) sof.setOrderRank(thisSo.getFields().size());
    else{
        sof.setOrderRank(0);
        thisSo = new SortOrder();
    }

    sof.setSortDirection("DESC");
    thisSo.addField(sof);
    Z.probe("RPT:!!!!!!ADDED DEFAULT SORT ORDER ----- ID DESC" );
}
else thisSo = this.so;

Iterator soi = thisSo.getFields().keySet().iterator();

boolean useOJRelease = false;
boolean useOJModule = false;
boolean useOJRelationshipGroup = false;
boolean useOJ = false;

prSorts.clear();

while (soi.hasNext()){
    //Z.probe("pre-sort-order ddeTable / itemType is " + ddeTable + " /
" + ddeItemId);

    key = (String)soi.next();
    sof = thisSo.getField(key);
    thisDDE = (DDEntry)(sortFields.get(key));
    if (thisDDE == null) continue;
    if (thisDDE.getTableName() != null)
        tmpTable = thisDDE.getTableName();
    if (this.hr) tmpTable = getHistoryEquivalent(tmpTable);
    if (thisDDE.getItemId() != null)
        tmpItemId = thisDDE.getItemId();
    else tmpItemId = "";

    if ("PROBLEM_RELEASE".equalsIgnoreCase(tmpTable)
        || "1".equalsIgnoreCase(tmpItemId)){
        prSorts.put(thisDDE.getName(),thisDDE);
        continue;
        tmpColumn = thisDDE.getColumnName();
        if (!addReleaseSql){
```

//

```

Report.java
addReleaseSql = true;
if ( prFilters.get(tmpName) == null && prSorts.get(tmpName)
== null )
    prSql.append(", i."+tmpColumn + " ");
}
}
if (thisDDE.getColumnName() != null)
    tmpColumn = TextManager.replace( thisDDE.getColumnName(), "||",
z.dbms.cat() );
else tmpColumn = "";
if ( tmpTable.startsWith("PROBLEM_RELEASE")
    && ("RELEASE_FOUND".equalsIgnoreCase(tmpColumn)
    || "PRODUCT_NAME".equalsIgnoreCase(tmpColumn)) ){
    useOJRelease = true;
}
else if ( tmpTable.startsWith("PROBLEM_MODULE")
    && "MODULE_ID".equalsIgnoreCase(tmpColumn) ){
    useOJModule = true;
}
else if ( tmpTable.startsWith("RELATIONSHIP_GROUP")
    && ("PARENT_PROBLEM_ID".equalsIgnoreCase(tmpColumn)
    || "RELATIONSHIP_GROUP_ID".equalsIgnoreCase(tmpColumn)) ){
    useOJRelationshipGroup = true;
}
}
} // end pre-sort order

```

```

boolean isReleaseItemTypeSet = false;

```

```

// finish getting sort order info

```

```

//*****

```

```

re = new ReportElement(null, oil, isChart, hr);
re.toLog(Z.log.DEBUG5);

```

```

boolean skipPreCondition = false;
boolean doCaseInsensitive = this.isCaseInsensitive;

```

```

if (!"SYSTEM".equals(su.getId()) && !"ADMIN".equals(su.getId())) {
    if (doCaseInsensitive) preCondition.append(" UPPER(" + re.getItem() +
".PRIVACY) = UPPER(?) ");
    else preCondition.append(" " + re.getItem() + ".PRIVACY = ? ");
    paramsPreCondition.add("PUBLIC");
    typesPreCondition.add("STRING");
    if (doCaseInsensitive) preCondition.append(" OR UPPER(" +
re.getItem() + ".ORIGINATOR) = UPPER(?) ");
    else preCondition.append(" OR UPPER(" + re.getItem() + ".ORIGINATOR) =
UPPER(?) ");
    paramsPreCondition.add(su.getId());
    typesPreCondition.add("STRING");
    if (doCaseInsensitive) preCondition.append(" OR UPPER(" +
re.getItem() + ".ASSIGNED_TO) = UPPER(?) ");
    else preCondition.append(" OR " + re.getItem() + ".ASSIGNED_TO = ? ");
    paramsPreCondition.add(su.getId());
    typesPreCondition.add("STRING");
}

```

```

Report.java
    if (doCaseInsensitive) preCondition.append(" OR UPPER(" +
re.getItem() + ".OWNER) = UPPER(?) ");
    else preCondition.append(" OR " + re.getItem() + ".OWNER = ? ");
    paramsPreCondition.add(su.getId());
    typesPreCondition.add("STRING");

    if
("YES".equalsIgnoreCase(Z.appDefaults.getAttribute("ENABLE_COMPANY_NAME_ACCESS"))) {
        String companyName = su.getCompanyName();

        // If my Company_Name from Security_User = Company_Name from
Application Default...
        // or if my Company_Name is null then I should be able to see
all records,
        // regardless of PRIVACY.

        //Z.probe(" companyName =" +companyName);
        //Z.probe("Z.appDefaults.getAttribute(COMpany_NAME) =" +
Z.appDefaults.getAttribute("COMPANY_NAME"));
        if (TextManager.isStringVisible(companyName)) {
            if
(companyName.equals(Z.appDefaults.getAttribute("COMPANY_NAME"))) {
                skipPreCondition = true;
            }
            else {
                preCondition.append(" OR exists (select 1 from ITEM p,
SECURITY_USER su " +
                    " where su.SECURITY_USER_ID = p.ORIGINATOR "
+
                    " and p." + PROBLEM_KEY + " = " +
re.getDriverAlias() + "." + PROBLEM_KEY);

                if (doCaseInsensitive)
                    preCondition.append(" and UPPER( su.COMPANY_NAME )
= UPPER(?) ) ");
                else
                    preCondition.append(" and su.COMPANY_NAME = ? )
");

                paramsPreCondition.add(companyName);
                typesPreCondition.add("STRING");
            }
        }
        else {
            skipPreCondition = true;
        }
    }

    if
("YES".equalsIgnoreCase(Z.appDefaults.getAttribute("ENABLE_PRIVACY_GROUPS"))) {
        su.setConnection(conn);
        ArrayList pg = su.getPrivacyGroups();

        //Z.probe("This users privacy group is null = " + (pg==null));
        if (pg != null) {

            //Z.probe("This users privacy group size is = " +
pg.size());
            if (pg.size() > 0 && pg.size() < 20) {

                if (doCaseInsensitive) preCondition.append(" OR UPPER(" +
re.getItem()+".PRIVACY ) in (" );

```



```

Report.java
else preCondition.append(" OR " + re.getItem()+" .PRIVACY
in ("");

int k = 0;
for (k = 0; k < pg.size(); k++) {
    if (k > 0) preCondition.append(", ");
    preCondition.append("?");
    if (doCaseInsensitive)
preCondition.append("UPPER(?)");
    paramsPreCondition.add((String) pg.get(k));
    typesPreCondition.add("STRING");
}
preCondition.append(")");
}
else if (pg.size() > 0) {
    preCondition.append(" OR " + re.getItem() + ".PRIVACY in
" +
        "( select PRIVACY_GROUP_ID " +
        " from PRIVACY_GROUP_USER where SECURITY_USER_ID
= ? )");
    paramsPreCondition.add((String) su.getId());
    typesPreCondition.add("STRING");
}
} // end enable privacy groups
} // end system and admin

if (skipPreCondition) {
    preCondition.setLength(0);
    paramsPreCondition.clear();
    typesPreCondition.clear();
}

// LOOP THROUGH THE FILTERS AND BUILD THE QUERY STRING
//-----
boolean isProblemItemTypeSet = false;

int timeSeriesDateParamPositions[] = {0,0}; // what is this ???
Iterator fi = filters.keySet().iterator();

Calendar startTimeSeriesCalendar = null;
Calendar endTimeSeriesCalendar = null;
String token = ""; // for use with ALLOWED_SEARCH_FILE_EXT
boolean hasNullToken = false;

String andOr = "";
String operator = "";

Filter tempFilter = null;

while (fi.hasNext()) {

    key = (String) fi.next();
    if ( key.indexOf("RANGE_START") > -1)
        baseName = key.substring(12);
    else if (key.indexOf("RANGE_STOP") > -1)
        baseName = key.substring(11);
    else
        baseName = key;

```

Report.java

```

tempFilter = (Filter)filters.get(key);
andOr = tempFilter.getAndOr();
operator = tempFilter.getOperator();

// Only process valid filter fields
//-----
if ((thisDDE = (DDEntry) filterFields.get(baseName)) == null)
continue;

re = new ReportElement(thisDDE, oil, isChart, hr);
re.toLog(Z.log.DEBUG5);

if (isChart){
    this.driverTable = "ITEM_HIST";
    re.setDdeTable("ITEM_HIST");
}
appendChartData = isChart;

if (!isReleaseItemTypeSet && addReleaseSql) {
    if (isChart) prSql.append(" from item_group ig, item_hist i
where i.item_type_id = 1 and i.item_id = ig.item1_id and i.change_type <> 'D' ");
    else prSql.append(" from item_group ig, item i where
i.item_type_id = 1 and i.item_id = ig.item1_id ");
    jd.addAll(prSql.toString(),
        "PR",
        jd.LEFT_JOIN,
        "ITEM2_ID",
        re.getDriverAlias(),
        DRIVER_KEY,
        1);

    isReleaseItemTypeSet = true;
    distinctTables.put("PR", thisDDE);
}

f = fg.getFilter(key);
rawfc = f.getFilterCriteriaResolved();
fc = new ArrayList();
criteriaSize = rawfc.size();

// debug check..
if (criteriaSize == 0) {
    ArrayList fcTest = f.getFilterCriteria();

    if (fcTest.size() > 0) Z.log.writeToLog(Z.log.ERROR, "RPT:
resolved mismatch error:" + fcTest);
}
hasNullCriteria = false;

// PREQUERY OPERATIONS - TEST FOR WILDCARS IN FILTER CRITERIA
//
-----
// if there is an "*" in the criteria, replace the wildcard with
DBMS.wildcard
// and do LIKE queries
wildcards = false;

```

```

Report.java
for (int k = 0; k < criteriaSize; k++) {
    rawValue = (String) rawfc.get(k);
    processedValue = TextManager.replace(rawValue, "*",
Z.dbms.WILDCARD());
    criteriaTableColumn.setLength(0);
    if ("{}null{}".equalsIgnoreCase(processedValue)) hasNullCriteria =
true;

    if ("USER".equalsIgnoreCase(re.getDdeDisplayType()))
        fc.add(processedValue.toUpperCase(Z.defaultLocale));
    else
        fc.add(processedValue);

    Z.log.writeToLog(Z.log.DEBUG, "RPT:raw/processed=" + rawValue +
"/" + processedValue);
    if (rawValue.indexOf("*") > -1) wildcards = true;
}

if ("DATE".equalsIgnoreCase(re.getDdeDisplayType())) {
    if (isTimeSeries){

        try{
            startTimeSeriesCalendar =
Convert.getCalendarFromStringWithMask(hasNullCriteria ? "" : startTimeSeriesDate,
dbt.getMaskFromDateFormat(su.getDateFormat())) );
        }
        catch(Exception e){
            startTimeSeriesCalendar =
Convert.getCalendarFromString(session,
hasNullCriteria ? "" : startTimeSeriesDate);
        }
        try{
            endTimeSeriesCalendar =
Convert.getCalendarFromStringWithMask(hasNullCriteria ? "" : stopTimeSeriesDate,
dbt.getMaskFromDateFormat(su.getDateFormat())));
        }
        catch(Exception e){
            endTimeSeriesCalendar =
Convert.getCalendarFromString(session,
hasNullCriteria ? "" : stopTimeSeriesDate);
        }
        dbt2.setUserNow(endTimeSeriesCalendar);
        dbtForTimeSeries.setUserNow(startTimeSeriesCalendar);
    }
    else startTimeSeriesCalendar =
Convert.getCalendarFromString(session, (String) (hasNullCriteria ? "" : fc.get(0))
);

    dbt.setUserNow(startTimeSeriesCalendar);

    if (fc.size() > 1)
        try{

            dbt2.setUserNow(Convert.getCalendarFromStringWithMask((String) (hasNullCriteria ? ""
: fc.get(1)),
dbt.getMaskFromDateFormat(su.getDateFormat())));
        }
        catch(Exception e){
            dbt2.setUserNow(Convert.getCalendarFromString(session,
(String) (hasNullCriteria ? "" : fc.get(1))));
        }
    }
}

```

```

    }
}

/*****
 * PROCESS KEYWORD -- NEVER FOR A HISTORY REPORT -- CRAZY
 * -----
 */
// TO DO -- search release level text
//-----
    if ("KEYWORD".equalsIgnoreCase(re.getDdeName())) {
        andKeyword.append(andOr + " ( ");
        andKeyword.append("( ");

// Look in the short_desc
//-----
        for (int k = 0; k < criteriaSize; k++) {
            if (k > 0) andKeyword.append(" and ");
            andKeyword.append("UPPER(" + re.getItem() + ".short_desc"
                + " LIKE '" + Z.dbms.WILDCARD() + "' " +
                + " UPPER(?) " + Z.dbms.cat() + " '" +
                Z.dbms.WILDCARD() + "' ");
            paramsKeyword.add((String) fc.get(k));
            typesKeyword.add("STRING");
        }
        andKeyword.append(") ");
        andKeyword.append(" or ");

// Look in the problem_text
//-----
        andKeyword.append("exists ( ");
        andKeyword.append(" select " + keywordHint + " 'x' from " +
            re.getItem_TEXT() +
            " where " + re.getItem_TEXT() + "." +
            PROBLEM_KEY +
            " = " + re.getItem() + "." + PROBLEM_KEY + "
        and ");
        andKeyword.append(" ( ");

        for (int k = 0; k < criteriaSize; k++) {
            if (k > 0) andKeyword.append(" and ");
            andKeyword.append("UPPER(" + re.getItem_TEXT() + ".text) ");
            andKeyword.append(" LIKE '" + Z.dbms.WILDCARD() + "' " +
                + " UPPER(?) " + Z.dbms.cat() + " '" +
                Z.dbms.WILDCARD() + "' ");
            paramsKeyword.add((String) fc.get(k));
            typesKeyword.add("STRING");
        }
        andKeyword.append(") ");
        andKeyword.append(") ");

        andKeyword.append(" or ");

// Look in the attachment desc
//-----
        andKeyword.append(" exists ( ");
        andKeyword.append(" select 'x' from " + re.getATTACHMENT() +
            " where " + re.getATTACHMENT() + ".item_id = "

```

```

Report.java
+ re.getItem() + "." + PROBLEM_KEY + " and ";
    andKeyword.append(" ( ");
    for (int k = 0; k < criteriaSize; k++) {
        if (k > 0) andKeyword.append(" and ");
        andKeyword.append("UPPER(" + re.getAttachment() +
".FILE_DESC) ");
        andKeyword.append(" LIKE '" + Z.dbms.WILDCARD() + "' " +
Z.dbms.cat()
        + " UPPER(?) " + Z.dbms.cat() + " '" +
Z.dbms.WILDCARD() + "' ");
        paramsKeyword.add((String) fc.get(k));
        typesKeyword.add("STRING");
    }
    andKeyword.append(") ");
    andKeyword.append(") ");

// Look in the BLOB
//-----
    if (this.searchAttach) {
        andKeyword.append("or exists ( ");
        andKeyword.append(" select 'x' " +
            " from attachment a, attachment_content ac " +
            " where ac.attachment_id = a.attachment_id " +
            " and a.item_id = " + re.getItem() + "." +
PROBLEM_KEY +
            " and a.content_type in (");
        String okExtensions =
Z.appDefaults.getAttribute("ALLOWED_ATTACH_SEARCH_FILE_EXT");

        StringTokenizer st = new StringTokenizer(okExtensions, ",");

        ArrayList orderedColumnNames = new ArrayList();
        int cnt = 0;

        while (st.hasMoreTokens() || cnt == 0) {
            cnt++;
            if (okExtensions.indexOf(",") == -1) {
                paramsKeyword.add(okExtensions);
                typesKeyword.add("STRING");
                andKeyword.append("?");
            }
            else {
                if (cnt > 1) andKeyword.append(",");
                token = (String) st.nextToken();
                if ("\\\\".equalsIgnoreCase(token)) {
                    token = "";
                    hasNullToken = true;
                }
                paramsKeyword.add(token);
                typesKeyword.add("STRING");
                andKeyword.append("?");
            }
        }
        andKeyword.append(")");

        if (hasNullToken) andKeyword.append(" or a.content_type is
null ");

        andKeyword.append(" and ");
        andKeyword.append(" ( ");
        for (int k = 0; k < criteriaSize; k++) {
            if (k > 0) andKeyword.append(" and ");
            BlobSearchInfo bsi = Z.dbms.blobInstr("ac.content_blob",
"?", (String) fc.get(k), loc);

```



```

Report.java
//filter criteria loop
for (int k = 0; k < criteriaSize; k++) {
    if (k == 0) andProblem.append(andOr + " ( ");
    else andProblem.append(" or ");
    andProblem.append("UPPER(" + re.getTAlias() + "." +
re.getDdeColumn() + ") ");
    andProblem.append(" LIKE UPPER(?) ");
    paramsProblem.add((String) fc.get(k));
    typesProblem.add("STRING");
}
andProblem.append(") ");
}

// Process wildcard Non-Driver Table Fields - ie in Problem
Release or Problem Module
// - build an exists clause for these

//-----
-----
else {
    if ((andExists = (StringBuffer)
existClauses.get(re.getDdeTable())) == null) {
        andExists = new StringBuffer();
        paramsExists = new ArrayList();
        typesExists = new ArrayList();
        if (re.getHasGrandParentTable()) { // this is never a
history reporting thing
            // also never a view thing
            andExists.append(" exists (select 1 from " +
re.getTAlias() +
            " where " + re.getTAlias() + "." +
re.getDdeChildKey() +
            " = " + re.getPtAlias() + "." +
re.getDdeParentKey() +
            " and " + re.getGptAlias() + "." +
re.getDdeGrandParentKey() +
            " = " + re.getPtAlias() + "." +
re.getDdeGrandChildKey());
        } else {
            andExists.append(" exists (select 1 from " +
re.getTAlias() +
            " where " + re.getTAlias() + "." +
re.getDdeChildKey() +
            " = " + re.getPtAlias() + "." +
re.getDdeParentKey() + " )");
        }
    } else {
        paramsExists = (ArrayList)
existParams.get(re.getTAlias());
        typesExists = (ArrayList)
existTypes.get(re.getTAlias());
    }

    //filter criteria loop
    for (int k = 0; k < criteriaSize; k++) {
        if (k == 0) andExists.append(" and ( ");
        else andExists.append(" or ");
        andExists.append("UPPER(" + re.getTAlias() + "." +
re.getDdeColumn() + ") ");
        andExists.append(" LIKE UPPER(?) ");
        paramsExists.add((String) fc.get(k));
    }
}

```



```

Report.java
typesExists.add("STRING");
}
if (criteriaSize > 0) andExists.append(" ");
andExists.append(" ");

/*
// deal with release level UDFs --- special case
//-----
if ("UDF".equalsIgnoreCase(ddeType) &&
!("1".equalsIgnoreCase(ddeItemId))) {
andUDF.append(" and exists ");
andUDF.append(" (select 1 " +
" from " + ITEM_UDF +
" where " + ITEM_UDF + "." + problemKey + " = " + ITEM +
"." + problemKey +
" and " + ITEM_UDF + ".udf_id = ? ");
paramsUDF.add(((Udf) udfs.get(ddeName)).getUdfId());
typesUDF.add("STRING");

//filter criteria loop
for (int k = 0; k < criteriaSize; k++) {
if (k == 0) andUDF.append(" and ");
else andUDF.append(" or ");
andUDF.append(" UPPER(" + ITEM_UDF + ".");
if ("NUMBER".equalsIgnoreCase(ddeDisplayType)) {
andUDF.append("value_number");
} else {
andUDF.append("value");
}
andUDF.append(") ");
andUDF.append(" LIKE UPPER(?) ");
paramsUDF.add((String) fc.get(k));
typesUDF.add("STRING");
}
andUDF.append(") ");
}
*/

existClauses.put(re.getTAlias(), andExists);
existParams.put(re.getTAlias(), paramsExists);
existTypes.put(re.getTAlias(), typesExists);
existAndOrs.put(re.getTAlias(), andOr);

Z.probe("RPT:wildcard>>>>existClauses " + existClauses);
Z.probe("RPT:wildcard>>>>paramsExists " + paramsExists);
}
} // end if wildcards

else {
/*****
* Not using wildcards
* -----
*/

// Process the UDFs
//-----
if ("UDF".equalsIgnoreCase(re.getDdeType())) {
//Z.probe("RPT:IN UDF ---- " + ddeName);

```

```

Report.java
// if --none-- is the selected value
if (hasNullCriteria && criteriaSize == 1) {
    andUDF.append(andOr + " not exists ");
    andUDF.append(" (select 1 " +
        " from " + re.getItem_UDF() +
        " where " + re.getItem_UDF() + "." +
PROBLEM_KEY +
        " =" + re.getItem() + "." + PROBLEM_KEY);
    if (this.hr) {
        andUDF.append(" and " + re.getItem_UDF() + "." +
getHistKey(re.getItem_UDF(), true) +
        " =" + re.getItem() + "." +
getHistKey(re.getItem(), false));
    }
    andUDF.append(" and " + re.getItem_UDF() + ".udf_id
= ? ) ");
    paramsUDF.add((Udf)
udfs.get(re.getDdeName())).getUdfId());
    typesUDF.add("STRING");
}
else {
    andUDF.append(andOr + " exists ");
    if (hasNullCriteria) andUDF.append(" ( ");
    andUDF.append(" (select 1 " +
        " from " + re.getItem_UDF() +
        " where " + re.getItem_UDF() + "." +
PROBLEM_KEY +
        " =" + re.getItem() + "." + PROBLEM_KEY);
    // if (this.hr) {
    // andUDF.append(" and " + ITEM_UDF + "." +
getHistKey(ITEM_UDF, true) +
    //
    // " =" + ITEM + "." + getHistKey(ITEM, false));
    //}
    andUDF.append(" and " + re.getItem_UDF() + ".udf_id
= ? ");
    paramsUDF.add((Udf) udfs.get(baseName)).getUdfId());
    typesUDF.add("STRING");

    //TODO: - make this a method
    // test for UDF Type -- this determines what field is
matched

    if ("LIST".equalsIgnoreCase(re.getDdeDisplayType())
        || "POPUP".equalsIgnoreCase(re.getDdeDisplayType())
        || "TAB".equalsIgnoreCase(re.getDdeDisplayType()))
        criteriaTableColumn.append(re.getItem_UDF() +
".udf_list_id");
    else if
("DATE".equalsIgnoreCase(re.getDdeDisplayType()))
        criteriaTableColumn.append(re.getItem_UDF() +
".value_date");
    else if
("NUMBER".equalsIgnoreCase(re.getDdeDisplayType()))
        criteriaTableColumn.append(re.getItem_UDF() +
".value_number");
    else{
        if (doCaseInsensitive)
criteriaTableColumn.append("UPPER("+re.getItem_UDF()+ ".value)");
        criteriaTableColumn.append(re.getItem_UDF()+
".value");
    }
}

```

```

Report.java
//      if (hasNullCriteria)
//          andUDF.append(" and (" + criteriaTableColumn);
//      else
//          andUDF.append(" and " + criteriaTableColumn);
//
//      if ("DATE".equalsIgnoreCase(re.getDdeDisplayType())) {
//
// do between
//
//          if (criteriaSize > 1) {
//              andUDF.append(" between ? and ? )");
//              paramsUDF.add(dbt.getDbNowTimestamp());
//              typesUDF.add("DATE");
//              paramsUDF.add(dbt2.getDbThenTimestamp(1));
//              typesUDF.add("DATE");
//          }
//          else {
//              if (re.getDdeName().indexOf("START") > -1) {
//                  andUDF.append(" >= ? )");
//                  paramsUDF.add(dbt.getDbNowTimestamp());
//                  typesUDF.add("DATE");
//              }
//              else if (re.getDdeName().indexOf("STOP") > -1) {
//                  andUDF.append(" <= ? )");
//
//              paramsUDF.add(dbt.getDbThenTimestampLessOneMinuete(1));
//                  typesUDF.add("DATE");
//              }
//              else {
//                  andUDF.append(" between ? and ? )");
//
//              paramsUDF.add(dbt.getDbThenTimestampLessOneMinuete(1));
//                  typesUDF.add("DATE");
//              }
//          }
//      }
//      else {
// build the string for multiple criteria values
//          andUDF.append(" in ( ");
//
// filter criteria loop
//          for (int k = 0; k < criteriaSize; k++) {
//              if (hasNullCriteria) {
//                  if ("{"null}".equalsIgnoreCase((String)
// fc.get(k))) {
//                      skipComma = true;
//                      continue;
//                  }
//              }
//              if (k > 0 && !skipComma) andUDF.append(", ");
//              skipComma = false;
//              if (doCaseInsensitive && !(
// "LIST".equalsIgnoreCase(re.getDdeDisplayType())
// || "POPUP".equalsIgnoreCase(re.getDdeDisplayType())
// || "TAB".equalsIgnoreCase(re.getDdeDisplayType())
// || "NUMBER".equalsIgnoreCase(re.getDdeDisplayType())) ){
//                  andUDF.append("UPPER(?)");
//              }

```

```

        Report.java
        else andUDF.append("?");

        paramsUDF.add((String) fc.get(k));
        typesUDF.add("STRING");
    }
    andUDF.append(" "); //end in criteria
    andUDF.append(" "); // end exists

    if (hasNullCriteria){
// THE OLD WAY
//andUDF.append(" or " + criteriaTableColumn + " is null ) "); //end or is null

        andUDF.append(" or not exists "); // car mod
        andUDF.append(" (select 1 " +
            " from " + re.getItemUDF() +
            " where " + re.getItemUDF() +
            re.getItem() + ".item_id ");
        if (this.hr){
            andUDF.append(" and " + re.getItemUDF() +
                " = " + re.getItem() + "." +
                getHistKey(re.getItemUDF(),true)+
                getHistKey(re.getItem(),false));
        }
        andUDF.append(" and " + re.getItemUDF() +
            //andUDF.append(" ) "); // car mod
            paramsUDF.add( (
            (Udf)udfs.get(re.getDdeName()) ).getUdfId() );
            typesUDF.add("STRING");
        }
    }
} // end test for {null}
} // end test for UDFs

// Process Driver Table Fields
//-----
        else if (driverTable.equalsIgnoreCase(re.getDdeTable())
            || (isChart
&&("STATUS_HIST".equalsIgnoreCase(re.getDdeTable()) ||
            "ITEM_HIST".equalsIgnoreCase(re.getDdeTable()) ) )
            || (isChart &&
"ITEM".equalsIgnoreCase(re.getDdeTable())
            || "PR".equalsIgnoreCase(re.getItem())
        {

            Z.log.writeToLog(Z.log.DEBUG4,"~~@~~RPT:Processing Driver
Table Fields ---- " +re.getDdeTable()+ " for DDEntry " + re.getDdeName());
//Z.probe("RPT:IN DRIVER TABLE ---- " + ddeName);

            if ("STATUS_HIST".equalsIgnoreCase(re.getDdeTable())){
                if (distinctTables.get(re.getTAlias()) == null){
                    distinctTables.put(re.getTAlias(), thisDDE);

                    fromCnt++;

jd.addAll(re.getDdeTable(),re.getTAlias(),jd.INNER_JOIN,re.getDdeChildKey(),re.getPt
Alias(),re.getDdeParentKey(),1);
                }
            }
        }
    }
}

```



```

Report.java
        andProblem.append(" >= ? ");
        paramsProblem.add(dbt.getDbNowTimestamp());
        typesProblem.add("DATE");
    }
    else if (re.getDdeName().indexOf("STOP") > -1) {
        andProblem.append(" <= ? ");
        if (isTimeSeries)
timeSeriesDateParamPositions[0] = paramsProblem.size();
        paramsProblem.add(dbt.getDbThenTimestampLessOneMinuete(1));
        typesProblem.add("DATE");
    }
    else {
        andProblem.append(" between ? and ? ");
        paramsProblem.add(dbt.getDbNowTimestamp());
        typesProblem.add("DATE");
    }
    paramsProblem.add(dbt.getDbThenTimestampLessOneMinuete(1));
    typesProblem.add("DATE");
}
}

// This joins in the item to item_hist to status_hist
//-----
        if (appendChartDate){
            appendChartDate = false;
//
//            if (doHistJoin){
//                andProblem.append(" and " + driverAlias + "."
+ driverKey + " = sh."+driverKey);
//                andProblem.append(" and " + tAlias + "." +
driverKey + " = item_hist." + driverKey);
//                andProblem.append(" and " + tAlias +
".timestamp = item_hist.timestamp ");
//                andProblem.append(" and item_hist.change_type
<> 'D' ");
//                doHistJoin = false;
//            }
//
        if (this.isStatusOnDate() || this.isStatusByDate()){
            andProblem.append(" and " + re.getTAlias() + "."
+ re.getDdeColumn() + " ");
            andProblem.append(" = ");
            andProblem.append(" (select MAX(x.timestamp) " +
                " from status_hist x " +
                " where x.item_id = " +
re.getDriverAlias() + ".item_id");
//
//            if (this.isStatusOnDate()){
//                Track where the parameters that will be replaced in the timeseries loop
//-----
                andProblem.append(" and trunc(x.timestamp) <= ?
) ");
                if (isTimeSeries)
timeSeriesDateParamPositions[1] = paramsProblem.size();
//{
//    timeSeriesDates.put(new Integer(paramsProblem.size()),
//                        dbt );
//}

```

Report.java

```

paramsProblem.add(dbt.getDbThenTimestampLessOneMinuete(1));
typesProblem.add("DATE");
//
//
//
between ? and ? ) ");
//
paramsProblem.add(dbt.getDbNowTimestamp());
typesProblem.add("DATE");
//
//
paramsProblem.add(dbt2.getDbThenTimestamp(1));
typesProblem.add("DATE");
//
//
}
else if (isProductOrReleaseChart){
    andProblem.append(" and not exists ");
    andProblem.append(" (select 1 " +
        " from status_hist x " +
        " where x.item_id = sh.item_id " +
        " and nvl(x.status, '-') <>
nvl(sh.status, '-') " +
        " and x.timestamp between sh.timestamp
and ? ) ");
        if (isTimeSeries)
timeSeriesDateParamPositions[1] = paramsProblem.size();
paramsProblem.add(dbt.getDbThenTimestampLessOneMinuete(1));
typesProblem.add("DATE");

//*****
// SPECIAL CASE FOR THE HISTORICAL STATUS BEING A FILTER
//

        } // end if product or release chart

//
//
//
= sh.item_id " );
//
//
= sh.item_id " );

        } // end if appendChartDate
    } // end if date is display type
else {
// build the string for multiple criteria values
    andProblem.append(" in (");
    ArrayList areaToAdd = new ArrayList();
    for (int k = 0; k < criteriaSize; k++) {
        if (hasNullCriteria) {
            if ("{"null}".equalsIgnoreCase((String)
fc.get(k))) {
                skipComma = true;
                continue;
            }
        }
        if (k > 0 && !skipComma) andProblem.append(", ");
        skipComma = false;
        if (doCaseInsensitive)

```



```

Report.java
andProblem.append("UPPER(?)");
else andProblem.append("?");

// include area in check
if (re.getDdeName().equals("PROJECT")
    && (Project.getAreaFromQualifiedProject((String)
fc.get(k)) != null)) {

areaToAdd.add(Project.getAreaFromQualifiedProject((String) fc.get(k)));
paramsProblem.add(Project.getProjectFromQualifiedProject((String) fc.get(k)));
}
else {
    paramsProblem.add((String) fc.get(k));
}
typesProblem.add("STRING");
}
andProblem.append(") "); // end in

// qualified areas
//-----
if (areaToAdd.size() > 0) {
    andProblem.append(" and " + re.getTAlias() +

".area_id in ( ");

    for (int k = 0; k < areaToAdd.size(); k++){
        if (k > 0) andProblem.append(",");
        andProblem.append("?");

        paramsProblem.add((String)areaToAdd.get(k));
        typesProblem.add("STRING");
    }
    andProblem.append(") ");
}

if (hasNullCriteria) andProblem.append(" or " +
criteriaTableColumn + " is null "); // end or is null
}

}

/*****
* Process non-problem table fields
* (like problem_release or problem_module) then build an exists
clause
*
* To Do: Note that we currently only go one level out from the
* driver table so that the relationships are always to
* problem.problem_id -- future changes may include a
* User Defined Table structure n levels deep
*/
else {
    Z.probe("RPT: Processing non-problem table field where
ddeName is " + re.getDdeName() + " ddeTable is " + re.getDdeTable() + " and
ddeColumn is " + re.getDdeColumn() + " has null Criteria = " + hasNullCriteria);

// determine the use of not exist clauses
useNotExists = false;
if (hasNullCriteria && criteriaSize == 1) {

```

```

Report.java
    if ( (re.getDdeTable().startsWith("PROBLEM_RELEASE")
        || re.getDdeTable().startsWith("ITEM_RELEASE"))
    )
        &&
        ("RELEASE_FOUND".equalsIgnoreCase(re.getDdeColumn()))
        &&
        ("PRODUCT_NAME".equalsIgnoreCase(re.getDdeColumn())) ) {
            useNotExists = true;
        }
        else if ((re.getDdeTable().startsWith("PROBLEM_MODULE")
            || re.getDdeTable().startsWith("ITEM_MODULE"))
            &&
            "MODULE_ID".equalsIgnoreCase(re.getDdeColumn())) {
                useNotExists = true;
            }
            else if
            (re.getDdeTable().startsWith("RELATIONSHIP_GROUP")
                &&
                "PARENT_ITEM_ID".equalsIgnoreCase(re.getDdeColumn())
                &&
                "RELATIONSHIP_GROUP_ID".equalsIgnoreCase(re.getDdeColumn())) {
                    useNotExists = true;
                }
            }
        if (useNotExists && criteriaSize > 1){
            hashCode = re.getTAlias() + re.getDdeColumn();
            orExists.put(hashCode,hashCode);
        }
        else hashCode = re.getTAlias();

//*****
//
// THIS IS A SPECIAL CASE FOR NON_NULLABLE FIELDS OF JOINED TABLES
//
// DO NOT EXIST
//
//*****

        if (useNotExists) {
            if (notExistClauses.get(hashCode) == null) {
                notExists = new StringBuffer();
                //if (criteriaSize > 1) notExists.append(" or ");
                //else notExists.append(" and ");

                // skip this one for history -- there is no history table this far out
                if (re.getHasGrandParentTable()) {

                    // not true for the RELATIONSHIP_GROUP_VIEW
                    notExists.append(" not exists (select 1 from " +
                        re.getTAlias() +
                        " where " + re.getPtAlias() + "." +
                        re.getDdeParentKey() +
                        "= " + re.getTAlias() + "." +
                        re.getDdeChildKey() +
                        " and " + re.getGptAlias() + "." +
                        re.getDdeGrandParentKey() +
                        "=" + re.getPtAlias() + "." +
                        re.getDdeGrandChildKey());

```

```

Report.java
    }
    else {
        notExists.append(" not exists (select 1 from " +
re.getTAlias() +
re.getDdeChildKey() +
re.getDdeParentKey());
        " where " + re.getTAlias() + "." +
        "=" + re.getDriverAlias() + "." +

// if (this.hr) {
//     notExists.append(" and " + tAlias + "." + getHistKey(tAlias, false) +
//     "=" + driverTable + "." + getHistKey(driverAlias, false));
// don't need the not null condition
// because of the database constraint
//}

        }
        notExists.append(" and " + re.getTAlias() + "." +
re.getDdeColumn() + " is not null ");
        notExistClauses.put(hashKey, notExists);

//Z.probe (" CREATED NEW NON_EXIST CLAUSES :" + notExists);
    }

// DO EXIST
//-----
    else if (!useNotExists || (useNotExists && criteriaSize >
1)){

        if ((andExists = (StringBuffer)
existClauses.get(hashKey)) == null) {

            andExists = new StringBuffer();
            paramsExists = new ArrayList();
            typesExists = new ArrayList();

// skip this one for history -- there is no history table this far out

            if (re.getHasGrandParentTable()) {
// not true for the RELATIONSHIP_GROUP_VIEW
                andExists.append(" exists (select 1 from " +
re.getTAlias() + ", " +
re.getPtAlias() + "." +
re.getTAlias() + "." + re.getDdeChildKey() +
re.getDdeGrandParentKey() +
re.getDdeGrandChildKey());
                re.getPtAlias() + " where " +
                re.getDdeParentKey() + " = " +
                " and " + re.getGptAlias() + "." +
                "=" + re.getPtAlias() + "." +
            }
            else {
                andExists.append(" exists (select 1 from " +
re.getTAlias() +
re.getDdeChildKey() +
re.getDdeParentKey());
                " where " + re.getTAlias() + "." +
                "=" + re.getPtAlias() + "." +
            }
        }
    }
// if (this.hr) {

```

```

Report.java
// andExists.append(" and " + tAlias + "." + getHistKey(tAlias, false) +
//      "=" + driverAlias + "." + getHistKey(driverAlias, false));
//}

    }
    }
    else{
        if (andExists.toString().endsWith(""))
andExists.setLength(andExists.length()-1);
        paramsExists = (ArrayList)existParams.get(hashKey);
        typesExists = (ArrayList)existTypes.get(hashKey);
    }

        if (doCaseInsensitive &&
!"DATE".equalsIgnoreCase(re.getDdeDisplayType()))
            criteriaTableColumn.append("UPPER("+ re.getTAlias()
+ "." + re.getDdeColumn()+")");
        else
            criteriaTableColumn.append(re.getTAlias() + "." +
re.getDdeColumn());

        if (hasNullCriteria && criteriaSize > 1 &&
!useNotExists)
            andExists.append(" and (" + criteriaTableColumn + "
");
        else
            andExists.append(" and " + criteriaTableColumn );

        if (hasNullCriteria && criteriaSize == 1){
            andExists.append(" is null ");
        }

        else if
("DATE".equalsIgnoreCase(re.getDdeDisplayType())) {
// do between

            if (fc.size() > 1) {
                paramsExists.add(dbt.getDbNow());
                typesExists.add("DATE");
                paramsExists.add(dbt2.getDbThenTimestamp(1));
                typesExists.add("DATE");
                andExists.append(" between ? and ? ");
            }
            else {
                if (re.getDdeName().indexOf("START") > -1) {
                    andExists.append(" >= ? ");
                    paramsExists.add(dbt.getDbNow());
                    typesExists.add("DATE");
                }
                else if (re.getTAlias().indexOf("STOP") > -1) {
                    andExists.append(" <= ? ");
                }

                paramsExists.add(dbt.getDbThenTimestampLessOneMinuete(1));
                typesExists.add("DATE");
            }
            else {
                andExists.append(" between ? and ? ");

                paramsExists.add(dbt.getDbThenTimestampLessOneMinuete(1));
                typesExists.add("DATE");
            }
        }
    }

```

```

Report.java
    }
  } else {
    andExists.append(" in ( ");

//filter criteria loop
    for (int k = 0; k < criteriaSize; k++) {
      if (hasNullCriteria) {
        if ("{"null}".equalsIgnoreCase((String)
fc.get(k))) {
          skipComma = true;
          continue;
        }
      }
      if (k > 0 && !skipComma) andExists.append(", ");
      skipComma = false;
      if (doCaseInsensitive)
        andExists.append("UPPER(?)");
      else andExists.append("?");
      paramsExists.add((String) fc.get(k));
      typesExists.add("STRING");
    }
    andExists.append(") "); // end in
    if (hasNullCriteria)
      andExists.append(" or " + criteriaTableColumn +
" is null "); // end or is null
  }

// put the andExists clause back in the hashmap
// put the paramsOther list into the hashmap

    andExists.append(" )" );
    existClauses.put(hashKey, andExists);
    existParams.put(hashKey, paramsExists);
    existTypes.put(hashKey, typesExists);
    existAndOrs.put(hashKey, andOr);

//Z.probe("RPT:>>>>existClauses " + existClauses);
//Z.probe("RPT:>>>>paramsExists " + paramsExists);
  }
//Z.probe("RPT:>>>>notExistClauses " + notExistClauses);
  } // end process non-problem fields
} // end wildcard/no wildcard
} // END FILTER ITERATOR

// Make sure that the STATUS_HIST table and ITEM_HIST table are included in
// summary reports. These queries still join to status history
//-----

//TO DO: create a report DDE object rather than hardcode
//-----
    if (isChart){
      if (distinctTables.get("sh") == null){
        distinctTables.put("sh", thisDDE);

        fromCnt++;
      }

jd.addAll("STATUS_HIST", "sh", jd.INNER_JOIN, "ITEM_ID", "ITEM_HIST", "ITEM_ID", 1);
    }

  }

/*    if (isProductOrReleaseChart){

```

```

Report.java
tAlias = oil.getTableAlias("STATUS_HIST","STATUS_HIST");
if (distinctTables.get(tAlias) == null){
distinctTables.put(tAlias, (DDEntry)filterFields.get("STATUS_HIST") );
//if (fromCnt++ > 0) from.append(", ");
//from.append("STATUS_HIST " + tAlias);
jd.addAll("STATUS_HIST",tAlias,"","","","",0);
}
if (distinctTables.get("ITEM_HIST") == null){
distinctTables.put("ITEM_HIST", (DDEntry)filterFields.get("ITEM_HIST") );
//if (fromCnt++ > 0) from.append(", ");
//from.append("ITEM_HIST ");
jd.addAll("ITEM_HIST",getAlias("ITEM_HIST"),"","","","",0);
}
}
*/
/*****
* Process problem table fields
* for the FROM CLAUSE -- includes aliases for the ordered index
*
* To Do: Note that we currently only go one level out from the
* driver table so that the relationships are always to
* problem.problem_id -- future changes may include a
* User Defined Table structure n levels deep
*/

String[] taa = oil.getTableAliasArray();
//Z.probe("RPT : oil.getTableAliasArray() = " +taa);

//      StringBuffer andPr = new StringBuffer();
//      Z.log.writeToLog(Z.log.DEBUG3, "RPT: taa is:" + taa);

      if (taa != null && taa.length > 0) {
          for (int j = 0; j < taa.length; j++) {
              fromCnt++;
              if (j > 0){
//andPr.append(" and " + taa[j] + "." + problemKey + " = " + taa[j - 1] + "." +
problemKey);
//if (fromCnt++ > 0) from.append(", ");
//from.append(driverTable + " " + taa[j]);
jd.addAll(this.driverTable,
          taa[j],
          jd.INNER_JOIN,
          PROBLEM_KEY,
          taa[j-1],
          PROBLEM_KEY,
          0);

              }
              else{
jd.addAll(this.driverTable,
          taa[j],
          "",
          "",
          "",
          "",
          0);

              }

          }

//      Z.log.writeToLog(Z.log.DEBUG3, "RPT: - from append=" + driverTable
+ " " + taa[j]);

```

```

    }
    //andProblem.append(andPr);
    distinctTables.put(re.getDriverAlias(),
/*(DDEntry)filterFields.get("ID")*/new DDEntry());
}

// always include the driver table
else {
//     if (fromCnt++ > 0) from.append(", ");
//     from.append(driverTable + " " + driverAlias);
//     fromCnt++;
//     jd.addAll(this.driverTable,
//             re.getDriverAlias(),
//             "",
//             "",
//             "",
//             "",
//             0);

    Z.log.writeToLog(Z.log.DEBUG3, "RPT: * from append= " +
this.driverTable + " " + re.getDriverAlias());

    distinctTables.put(re.getDriverAlias(),
/*(DDEntry)filterFields.get("ID")*/new DDEntry());
}

//-----
Z.log.writeToLog(Z.log.DEBUG5, "RPT : before the sort -->
select.toString() : " +select.toString());
Z.log.writeToLog(Z.log.DEBUG5, "RPT : before the sort -->
from.toString() : " +from.toString());
Z.log.writeToLog(Z.log.DEBUG5, "RPT : before the sort -->
where.toString() : " +where.toString());
Z.log.writeToLog(Z.log.DEBUG5, "RPT : before the sort -->
andItemType.toString() : " +andItemType);
Z.log.writeToLog(Z.log.DEBUG5, "RPT : before the sort -->
preCondition.toString() : " +preCondition);
Z.log.writeToLog(Z.log.DEBUG5, "RPT : before the sort -->
andProblem.toString() : " +andProblem);
Z.log.writeToLog(Z.log.DEBUG5, "RPT : before the sort -->
andOther.toString : " + andOther);
Z.log.writeToLog(Z.log.DEBUG5, "RPT : before the sort -->
andUDF.toString() : " + andUDF);
Z.log.writeToLog(Z.log.DEBUG5, "RPT : before the sort -->
andKeyword.toString() : " +andKeyword);
Z.log.writeToLog(Z.log.DEBUG5, "RPT : before the sort -->
andSortOrder.toString() : " +andSortOrder);
Z.log.writeToLog(Z.log.DEBUG5, "RPT : before the sort -->
orderBy.toString() : " +orderBy);
Z.log.writeToLog(Z.log.DEBUG5, "RPT : before the sort -->
groupBy.toString() : " +groupBy);

//-----
/*****DOING SORT ORDER*****/
* PROCESSING THE SORT
* -----
*
* 1. Fields that are multi valued are ignored --
*     these should already be filtered by the UI
* 2. It is ASSUMED that sort orders will never
*     contain PROBLEM_RELEASE and PROBLEM_MODULE

```



```

                                Report.java
*      tables at the same time. Therefore we do not need to
*      do a distinct select clause. Again thes should be
*      already filtered by the UI
*
* Logic
* -----
*
*      - Iterate through the sort order fields and use a distinctTables HashMap
*      to filter unique tables for creating the join.
*      - Process the UDF as aliases UDF tables & generate alias to then be used in
the order_by
*      - Process any Lookups as special cases
*
*/

    HashMap ddeDisplayedDates = new HashMap();

    StringBuffer udfIndexHint = new StringBuffer();
    boolean anotherLookup = false;
    boolean doneFrom = false;
    String hasValueSufx = "N";

    int origSofCnt = thisSo.getFields().size();

    // add the calc fields for processing in the from clause
    // then remove them from the sort order
    HashMap calcDDNames = new HashMap();

    if (this.calculatedFields.size() > 0) {
        doCalcFields = true;
        CalculatedField cField = null;

        for (int j = 0; j < this.calculatedFields.size(); j++) {
            cField = (CalculatedField) this.calculatedFields.get(j);
            sof = new SortOrderField();
            sof.setDDName(cField.getDDName());
            sof.setOrderRank(j + origSofCnt);
            calcDDNames.put(cField.getDDName(), "");
            thisSo.getFields().put(cField.getDDName(), sof);
        }
    }

    // THIS IS TO TEST WHETHER WE OUTER JOIN in the summary report
    // we need to do outer join for not nullable fields in the join table
    // (ie count the number of null release found is an outer join
    //      but to count the number of null release fixed /or release
resolution,
    //      they need to be in a release to be counted)

    Iterator sofi = thisSo.getFields().keySet().iterator();

    //Z.log.writeToLog(Z.log.DEBUG5, "RPT: +++ thisSo.getFields().keySet()
+++ " + thisSo.getFields().keySet() );

    udfCnt = 0;
    sofCnt = 0;
    boolean isCalcField = false;
    String ojStr = "";
    String ojStr2 = "";

    String tempColumnName = "";

```

```

Report.java
String tempDriverTable = "";

if (isChart) tempDriverTable = "ITEM_HIST";
else tempDriverTable = driverTable;

Z.probe("RPT: < sort order loop > the tempDriverTable is : " +
tempDriverTable);
Z.probe("RPT: < sort order loop > prSorts : " + prSorts);

StringBuffer fromTable = new StringBuffer();

while (sofi.hasNext()) {
    sortOrderInfo = new SortOrderInfo();
    key = (String) sofi.next();

    Z.log.writeToLog(Z.log.DEBUG5, "RPT: < sort order loop > for key :
" + key);

    if (calcDDNames.get(key) != null) isCalcField = true;
    else isCalcField = false;

    sof = thisSo.getField(key);
    thisDDE = (DDEntry) (sortFields.get(key));

    if (thisDDE == null) {
        Z.log.writeToLog(Z.log.DEBUG5, "RPT:ALERT!!!!!!!!!! COULD NOT
FIND DDEntry FOR : " + key + " in sortfields ");
        continue;
    }

    re = new ReportElement(thisDDE, oil, isChart, hr);
    re.toLog(Z.log.DEBUG5);

    if (prSorts.get(re.getDdeName()) != null && !isSummaryReport){
        Z.log.writeToLog(Z.log.DEBUG5, "RPT:ALERT!!!!!!!!!! Skipping
SortOrder for DDEName (can't sort on releases) = " + re.getDdeName());
        continue; // skip releases... these get processed in the
details - in getSelect() method
    }

    // filter by item_type
    // CURRENTLY -- WE ALWAYS INCLUDE ITEM TABLE WITH TYPE 0
    //-----
    if (!isProblemItemTypeSet) {
        andItemType.append(" and " + re.getItem() + ".item_type_id = ?
");
        isProblemItemTypeSet = true;
        paramsItemType.add("0");
        typesItemType.add("STRING");
    }

    //-----DEBUG STUFF-----
    /*
Z.log.writeToLog(Z.log.DEBUG5, "~~~~~");
Z.log.writeToLog(Z.log.DEBUG5, "RPT~~~~sort order:ddeType = " +
re.getDdeType());
Z.log.writeToLog(Z.log.DEBUG5, "RPT~~~~ sort order:ddeDisplayType =
"+ re.getDdeDisplayType() );
Z.log.writeToLog(Z.log.DEBUG5, "RPT~~~~ sort order:ddeName = "
+re.getDdeName());

```

```

Report.java
    Z.log.writeToLog(Z.log.DEBUG5,"RPT~~~~~ sort order:ddeTable =" +
re.getDdeTable());
    Z.log.writeToLog(Z.log.DEBUG5,"RPT~~~~~ sort order:ddeColumn =" +
re.getDdeColumn());
    Z.log.writeToLog(Z.log.DEBUG5,"RPT~~~~~ sort order:ddeParentTable ="
+re.getDdeParentTable());
    Z.log.writeToLog(Z.log.DEBUG5,"RPT~~~~~ sort order:ddeParentKey =" +
re.getDdeParentKey() );
    Z.log.writeToLog(Z.log.DEBUG5,"RPT~~~~~ sort order:ddeChildKey ="
+re.getDdeChildKey() );
    Z.log.writeToLog(Z.log.DEBUG5,"RPT~~~~~ sort order:ddeLookupTable
="+re.getDdeLookupTable());
    Z.log.writeToLog(Z.log.DEBUG5,"RPT~~~~~ sort order:ddeLookupKey
="+re.getDdeLookupTable());
    Z.log.writeToLog(Z.log.DEBUG5,"RPT~~~~~ sort
order:ddeLookupColumn1="+re.getDdeLookupColumn1());
    Z.log.writeToLog(Z.log.DEBUG5,"RPT~~~~~ sort
order:ddeLookupColumn2="+re.getDdeLookupColumn2());
    Z.log.writeToLog(Z.log.DEBUG5,"RPT~~~~~ sort
order:ddeLookupColumn3="+re.getDdeLookupColumn3());
*/
//-----DEBUG STUFF-----

//*****
// * CREATING THE UDF JOINS
//

    fromTable.setLength(0);

    if ("UDF".equalsIgnoreCase(re.getDdeType())){
        tableAlias = re.getDdeName(); //"UDF" +
String.valueOf(udfCnt++);

        //This was for before we used views
        //-----
        //tableAlias = "IU_" + ddeName;
        //tableListAlias = "UL_" + ddeName;
        //tableUdfAlias = "U_" + ddeName;

        distinctTables.put(tableAlias, thisDDE);

        udfIndexHint.append(" " + Z.dbms.indexHintString(tableAlias,
"IX_ITEM_UDF1"));

        sortOrderInfo.setTableAlias(tableAlias);
        //sortOrderInfo.setTableListAlias(tableListAlias);
        sortOrderInfo.setSortOrderField(sof);
        sortOrderInfo.setSortBy(tableAlias + ".VALUE");
        sortOrderInfo.setSortByRawVal(tableAlias + ".VALUE");

        fromTable.append("(select distinct "+re.getItem_Udf()+" .item_id
\"ITEM_ID\", "); // need this for the bad_data problem

        if (this.hr){
            fromTable.append(re.getItem_Udf()+"."+getHistKey(re.getItem_Udf(),true) + " " +
getHistKey(re.getItem(),true)+", ");
        }

        if( "LIST".equalsIgnoreCase(re.getDdeDisplayType())
|| "POPUP".equalsIgnoreCase(re.getDdeDisplayType())
|| "TAB".equalsIgnoreCase(re.getDdeDisplayType())){

```

Report.java

```

        fromTable.append(" udf_list.title TITLE,
udf_list.udf_list_id VALUE, udf_list.sort_seq SORT_SEQ " +
        " from udf, udf_list, "+re.getItemUDF() +
        " where udf.udf_id = "+re.getItemUDF()+".udf_id " +
        " and udf_list.udf_list_id =
"+re.getItemUDF()+".udf_list_id " +
        " and udf.name = ? ) ");

        sortOrderInfo.setSortBy(tableAlias + ".TITLE");
        sortOrderInfo.setSortByRawVal(tableAlias + ".VALUE");

    }
    else if( "DATE".equalsIgnoreCase(re.getDdeDisplayType()) ){
        fromTable.append( re.getItemUDF()+ ".value_date VALUE " +
        " from udf, "+re.getItemUDF()+
        " where udf.udf_id = "+re.getItemUDF()+".udf_id " +
        " and udf.name = ? ) ");
    }
    else if( "NUMBER".equalsIgnoreCase(re.getDdeDisplayType()) ){
        fromTable.append( re.getItemUDF()+ ".value_number VALUE " +
        " from udf, "+re.getItemUDF()+
        " where udf.udf_id = "+re.getItemUDF()+".udf_id " +
        " and udf.name = ? ) ");
    }
    else if( "USER".equalsIgnoreCase(re.getDdeDisplayType()) ){
        sortOrderInfo.setSortBy(
this.getFormattedNameForQuery(tableAlias,
        "security_user_id",
        "first_name",
        "last_name") );

        fromTable.append(
        re.getItemUDF()+ ".value VALUE, " +
        "su.first_name FIRST_NAME, su.last_name LAST_NAME, "
+
        " su.security_user_id SECURITY_USER_ID " +
        " from security_user su, udf, "+re.getItemUDF()+
        " where udf.udf_id = "+re.getItemUDF()+".udf_id " +
        " and su.security_user_id = " + re.getItemUDF() +
".value " +
        " and udf.name = ? ) ");
    }
    else{
        fromTable.append( re.getItemUDF()+ ".value VALUE " +
        " from udf, "+re.getItemUDF()+
        " where udf.udf_id = "+re.getItemUDF()+".udf_id " +
        " and udf.name = ? ) ");
    }
    paramSortOrder.add(re.getDdeName());
    typeSortOrder.add("STRING");

    // join in the UDF from table
    //-----
    fromCnt++;
    jd.addAll(fromTable.toString(),
        tableAlias,
        jd.LEFT_JOIN,
        DRIVER_KEY,
        re.getDriverAlias(),
        DRIVER_KEY,

```

```

Report.java
1);

        if (this.hr) {
            andSortOrder.append(" and " + tableAlias + "." +
getHistKey(tableAlias,true) +
            Z.dbms.ojequals() + driverTable + "." +
getHistKey(driverTable,false));
        }

        soFields.put(String.valueOf(sof.getOrderRank()), sortOrderInfo);

    }

    /**
     * CREATING NON-UDF JOINS
     * -----
     *
     * NOTE: I only go ONE table out.  Cannot sort on a MULTIPLE field.
     * The sort fields that are more than one table out from the driver
table are ignored
     * for this reason
     *
     * Oct 2002 -- going to sort on child fields within the getselect()
method ...
     * I will pass on the stored sort orders for the multiple children
     * and process them accordingly within each one (currently I hard
code the sort on release_timestamp)
     *
     * If the field is a lookup, then process the logic for this as
well.
     *
     */
    else if (TextManager.isStringVisible(re.getDdeTable())) {

        useOJ = false;
        ojStr = Z.dbms.ojequals();
        if ( re.getDdeTable().startsWith("PROBLEM_RELEASE")) useOJ =
useOJRelease;
        else if ( re.getDdeTable().startsWith("PROBLEM_MODULE")) useOJ =
useOJModule;
        else if ( re.getDdeTable().startsWith("RELATIONSHIP_GROUP"))
useOJ = useOJRelationshipGroup;
        ojStr = useOJ ? Z.dbms.ojequals() : " = ";
        ojStr2 = useOJ ? jd.LEFT_JOIN : jd.INNER_JOIN;

        tempObject = distinctTables.get(re.getTAlias());
        if (tempObject != null){
            tempColumnName = ((DDEntry) tempObject).getColumnName();
            if (tempColumnName == null) tempColumnName = "";
            else tempColumnName = TextManager.replace(tempColumnName,
"||", Z.dbms.cat());
        }

        // Join in new table
        // -----
        // don't join in tables if it is a chart -- the sorted fields
are
        // selected from the multi-select box as filters -- thus, the
from clause is built

```

Report.java

```

    if (isProductOrReleaseChart){
        tableAlias = re.getTAlias();
    }
    else if ( tempObject == null ) {

        // Create the joins (here we only go 1 or 2 tables OUT)
        //-----
        if
(tempDriverTable.equalsIgnoreCase(re.getDdeParentTable())) {

            if (this.hr){
                andSortOrder.append(re.getTAlias() + "." +
getHistKey(re.getDdeTable(), false) +
getHistKey(re.getDdeTable(), false));
                ojStr + re.getPtAlias() + "." +
            }

            fromCnt++;
            jd.addAll(re.getDdeTable(),
                    re.getTAlias(),
                    ojStr2,
                    re.getDdeChildKey(),
                    re.getPtAlias(),
                    re.getDdeParentKey(),
                    1);

        }

        // not for history - doesn't go that far out
        // this doesn't happen for the view
        //-----
        else if (re.getHasGrandParentTable()) {

            jd.addAll(re.getDdeTable(),
                    re.getTAlias(),
                    ojStr2,
                    re.getDdeChildKey(),
                    re.getPtAlias(),
                    re.getDdeParentKey(),
                    1);

            jd.addAll(re.getDdeParentTable(),
                    re.getPtAlias(),
                    ojStr2,
                    re.getDdeGrandChildKey(),
                    re.getDdeGrandParentTable(),
                    re.getDdeGrandParentKey(),
                    2);

            distinctTables.put(re.getPtAlias(), thisDDE);
        }

        else if (tempDriverTable.equalsIgnoreCase(re.getTAlias())) {

            fromCnt++;
            jd.addAll(tempDriverTable,
                    re.getTAlias(),
                    "",
                    "",
                    "",
                    "",
                    1);
        }
    }

```

```

Report.java
    0);
}
tableAlias = re.getTAlias(); // returns driverAlias if none
found
    }
    // If the join is on the same field/ existing table, then we
    need to alias the table
    // but not a calculation on the same field (ie assigned_to /
    originator / owner
    //-----
    else if (re.getDdeColumn().equalsIgnoreCase( tempColumnName )
    &&
    TextManager.isStringInvisible(re.getDdeLookupColumn1())) {
        tableAliasCnt++;
        tableAlias = re.getTAlias() + "_" +
    String.valueOf(tableAliasCnt);

        // Create the joins (here we only go 1 or 2 tables out)
        // -----
        if
    (tempDriverTable.equalsIgnoreCase(re.getDdeParentTable())) {
            if (this.hr){
                andSortOrder.append(tableAlias + "." +
    getHistKey(tableAlias, true) +
                ojStr + re.getPtAlias() + "." +
    getHistKey(re.getDdeParentTable(), false));
            }

            fromCnt++;
            jd.addAl1(re.getDdeTable(),
                tableAlias,
                ojStr2,
                re.getDdeChildKey(),
                re.getPtAlias(),
                re.getDdeParentKey(),
                1);
        }

        // don't have any history tables this far out
        // don't worry about this with views
        else if (re.getHasGrandParentTable()) {
            jd.addAl1(re.getDdeTable(),
                tableAlias,
                ojStr2,
                re.getDdeChildKey(),
                re.getPtAlias(),
                re.getDdeParentKey(),
                1);

            jd.addAl1(re.getDdeParentTable(),
                re.getPtAlias(),
                ojStr2,
                re.getDdeGrandChildKey(),

```



```

Report.java
    re.getGptAlias(),
    re.getDdeGrandParentKey(),
    2);
}
}
else if (tempDriverTable.equalsIgnoreCase(re.getTAlias())) {
    fromCnt++;
    jd.addAll(tempDriverTable,
        re.getTAlias(),
        "",
        "",
        "",
        "",
        0);
}
else {
    tableAlias = re.getTAlias();
}
distinctTables.put(tableAlias, thisDDE);
sortOrderInfo.setSortOrderField(sof);
// Lookup is a calculation
// -----
if ((TextManager.isStringVisible(re.getDdeLookupColumn1()))
    && (TextManager.isStringInvisible(re.getDdeLookupKey()))) {
    sortOrderInfo.setTableAlias(""); // table/alias of sorted
field
sortOrderInfo.setSortBy(TextManager.replace(re.getDdeLookupColumn1(), "p.",
re.getDriverAlias() + "."));

sortOrderInfo.setSortByRawVal(TextManager.replace(re.getDdeLookupColumn1(), "p.",
re.getDriverAlias() + "."));
}

// Lookup is a Lookup
// -----
else if (TextManager.isStringVisible(re.getDdeLookupKey())) {
    anotherLookup = false;
    sortOrderInfo.setSortByRawVal(re.getDdeLookupKey());

    // the table's there in distinct tables as the ddeTable,
    // but the lookup is in the same table (ie ddeTable ==
ddeLookupTable)
    // so don't add it to the from and do not alias it, just
bring data back

// -----
    if
(re.getDdeTable().equalsIgnoreCase(re.getDdeLookupTable()))
    ||
TextManager.isStringInvisible(re.getDdeLookupTable())) {

```

```

Report.java
    luTableAlias = re.getDdeLookupTable();
}

// Join in new lookup table ....TO DO: LOOKUPS ON
RELATIONSHIP GROUP
//-----
    else if ((tempObject =
distinctTables.get(re.getDdeLookupTable())) == null) {
        luTableAlias = re.getDdeLookupTable();
        anotherLookup = true;
    }

// If the join is on the same field/ existing table, then
we need to alias the table
//-----
    else if
(re.getDdeLookupColumn1().equalsIgnoreCase(((DDEntry)
tempObject).getLookupColumn1())) {

        tableAliasCnt++;
        luTableAlias = re.getDdeLookupTable() + "_" +
String.valueOf(tableAliasCnt);

        Z.log.writeToLog(Z.log.DEBUG3, "RPT:from append here:if
the join is on the same field");
        anotherLookup = true;
    }

    if (anotherLookup) {
        distinctTables.put(luTableAlias, thisDDE);

        // DOING LOOKUP - 0/1 TABLES OUT -- already
        //-----
        if (re.getDdeTable().equalsIgnoreCase(tempDriverTable)
            ||
re.getDdeParentTable().equalsIgnoreCase(tempDriverTable)
            || (isChart) ) {

            fromCnt++;

            jd.addAll(re.getDdeLookupTable(),
                luTableAlias,
                jd.LEFT_JOIN,
                re.getDdeLookupKey(),
                ("ITEM".equalsIgnoreCase(re.getDdeTable()))
                ? re.getDriverAlias() : tableAlias ,
                re.getDdeColumn(),
                1);
        }
    }

sorted field
sortOrderInfo.setTableAlias(luTableAlias); // table name of

lookup - a user lookup // The column you sort on is a regular lookup or else a USER
is a concatenation
if ("USER".equalsIgnoreCase(re.getDdeDisplayType())) {
    sortOrderInfo.setSortBy(

```

```

Report.java
this.getFormattedNameForQuery(luTableAlias,
                                re.getDdeLookupKey(),
                                re.getDdeLookupColumn1(),
                                re.getDdeLookupColumn2()) );
    }
    else {
        sortOrderInfo.setSortBy(re.getDdeLookupColumn1());
    }
}
// Not a Lookup
else {
    sortOrderInfo.setTableAlias(tableAlias); // table
name/alias of sorted field
    sortOrderInfo.setSortBy(re.getDdeColumn());
    sortOrderInfo.setSortByRawVal(re.getDdeColumn());
}

//Z.probe(" ADDING SORT ORDER to soFields " + sof.getDDName() +
" " +sortOrderInfo[0]);
soFields.put(String.valueOf(sof.getOrderRank()), sortOrderInfo);
} // end if lookup not null
} // end while sofi hasNext()

// Always include the problem table
//-----
if ((tempObject = distinctTables.get(re.getDriverAlias())) == null) {
    fromCnt++;
    jd.addAll(tempDriverTable,
              re.getTAlias(),
              "",
              "",
              "",
              "",
              0);

    distinctTables.put(re.getDriverAlias(),
(DDEntry)filterFields.get(re.getDriverAlias()));
}

// remove the calcFields from the Sort Order Object - processing done
//-----
if (doCalcFields) {
    Iterator cdi = calcDDNames.keySet().iterator();

    while (cdi.hasNext())
        thisSo.getFields().remove(cdi.next());
}

// Build the from clause
//-----
jd.toLog(Z.log.DEBUG5);

String[] s = Z.dbms.buildFromClause(jd);

Z.probe("@@@@@@@@@@@@@@@@ QueryBuilder.buildFromClause(jd)[0] FROM
CLAUSE : "+ s[0]);
Z.probe("@@@@@@@@@@@@@@@@ QueryBuilder.buildFromClause(jd)[1] WHERE
CLAUSE : "+ s[1]);

```

Report.java

```

from.append(s[0]);
andSortOrder.append(s[1]);

/*****
 *   BUILD THE ORDER BY CLAUSE
 *   -----
 *   get from the soFields TreeMap sequentially using the key (this is a
rank)
 *   to retrieve sortOrderInfo
 */

String fieldAlias = "";
String sortByRawVal = "";
String rawValSelect = "";
String rawValGroupBy = "";
SortOrderInfo sofInfo = null;
StringBuffer sortSelect = new StringBuffer();
orderBy.append(" ORDER BY ");
boolean skipSort = false;

Iterator iter = soFields.keySet().iterator();
while (iter.hasNext()) {
    String keyValue = (String) iter.next();
    sofInfo = (SortOrderInfo) soFields.get(keyValue);
    temp = sofInfo.getTableAlias(); // the table alias
    sof = sofInfo.getSortOrderField(); // the sortOrderField
    fieldAlias = sof.getDDName();

    // don't add calculated fields to the select or group_by or order_by
    if (doCalcFields) {
        if (calcDDNames.get(fieldAlias) != null) {
            skipSort = true;
            //Z.probe("skipping group_by/ orderby processing for " +
fieldAlias);
            //Z.probe(" going to continue;");
            continue;
        }
    }

    thisDDE = (DDEntry) sortFields.get(fieldAlias);
    tmpDisplayType = thisDDE.getDisplayType();
    tmpType = thisDDE.getType();
    tmpName = thisDDE.getName();

    //Z.probe("Doing group by order by for dde " + fieldAlias);
    if (!keyValue.equals((String) soFields.firstKey()) && !skipSort) {
        orderBy.append(", ");
        skipSort = false;
    }
    if (this.isSummaryReport() && !skipSort) {
        if (!keyValue.equals((String) soFields.firstKey())) {
            groupBy.append(", ");
            gSelect.append(", ");
        }
        skipSort = false;
    }
}

```

Report.java

```

    }
    if ("".equalsIgnoreCase(temp)) {
        sortBy = sofInfo.getSortBy();
        sortByRawVal = sofInfo.getSortByRawVal();
    }
    else {
        if ("UDF".equalsIgnoreCase(tmpType)) {
            if ("LIST".equalsIgnoreCase(tmpDisplayType)
                || "POPUP".equalsIgnoreCase(tmpDisplayType)
                || "TAB".equalsIgnoreCase(tmpDisplayType)){
                sortBy =sofInfo.getTableAlias() + ".sort_seq, " +
sofInfo.getSortBy();
                sortByRawVal = sofInfo.getSortByRawVal();
            }
            else if ("USER".equalsIgnoreCase(tmpDisplayType))
                sortBy = sofInfo.getSortBy();
            else{
                sortBy = sofInfo.getSortBy();
                sortByRawVal = sofInfo.getSortByRawVal();
            }
        }
        else {
            // special case for users/ dates -- alias already done in
the lookup section
            if ("USER".equalsIgnoreCase(tmpDisplayType))
                sortBy = sofInfo.getSortBy();
            else{
                if (Z.dictionary.hasOrderRank(tmpName))
                    sortBy = temp + ".sort_seq, " + temp + "." +
sofInfo.getSortBy();
                else
                    sortBy = temp + "." + sofInfo.getSortBy();
            }
            if ("DATE".equalsIgnoreCase(tmpDisplayType))
                sortByRawVal = temp + "." + sofInfo.getSortBy();
            else
                sortByRawVal = temp + "." + sofInfo.getSortByRawVal();
        }
    }

    orderBy.append(sortBy + " " + sof.getSortDirection());
    if (this.isSummaryReport()) {
        if (TextManager.isStringVisible(sofInfo.getSortByRawVal())) {
            rawValSelect = ", " + sortByRawVal + " " + fieldAlias +
valuesuffix + " ";
            rawValGroupBy = ", " + sortByRawVal + " ";
            if ("DATE".equalsIgnoreCase(tmpDisplayType)) hasValueSuffix =
"D";
            else hasValueSuffix = "Y";
        }
        else {
            rawValSelect = "";
            rawValGroupBy = "";
            hasValueSuffix = "N";
        }
        groupBy.append(sortBy + rawValGroupBy);
        gSelect.append(sortBy + " " + fieldAlias + rawValSelect);
        groupByAliases.put(fieldAlias, hasValueSuffix);
    }

```

```

Report.java
        if ("DATE".equals(tmpDisplayType))
ddeDisplayedDates.put(fieldAlias, "");
    }
    else if (addReleaseSql){
        sortSelect.append(", " + sortBy /*+ " " + fieldAlias +
rawValselect*/);
    }
}
Z.log.writeToLog(Z.log.DEBUG5," DONE THE SORT ORDER LOGIC" );

/*****
 * PROCESS THE QUERY TO GET A LIST OF IDS
 *
 * 1) build query from constituent parts computed above
 * 2) bind params
 * 3) execute
 */

int gbSize = 0;
String hintString = Z.dbms.getOrderedHintString(oil.getIndexHintArray())
+ udfIndexHint.toString();
String iHint = Z.dbms.getHint(hintString);

select.append("select ");
select.append(iHint);
select.append(" ");

re = new ReportElement(null,oil,isChart, hr);
re.toLog(Z.log.DEBUG5);

if (this.isSummaryReport()) {
    gbSize = groupByAliases.size();

    if (!doCalcFields) {
        // if (this.hr) {
        //
        // select.append(" count(" +
/*"distinct"*/ "*) ");
        //
        // select.append(" count(");
        // select.append(driverAlias);
        // select.append(".");
        // select.append(driverKey);
        // select.append(") ");
        // select.append(idCountAlias);
        //
        // }
        // else {
        select.append(" count( distinct ");
        select.append(re.getDriverAlias());
        select.append(".");
        select.append(DRIVER_KEY);
        select.append(") ");
        select.append(idCountAlias);
        //
        // }
    }
    else {
        CalculatedField cField = null;
        StringBuffer tempAlias = new StringBuffer();

        for (int k = 0; k < this.calculatedFields.size(); k++) {
            if (k > 0) select.append(", ");

            cField = (CalculatedField) this.calculatedFields.get(k);
            tempAlias.setLength(0);

```

```

                                Report.java
tempAlias.append(cField.getDDName());
tempAlias.append("_");
tempAlias.append(cField.getCalculation());
groupByAliases.put(tempAlias.toString(), "CALC");

select.append(cField.getCalculation());
select.append("(");
if (!this.hr) select.append("distinct ");
select.append(cField.getDDName());
select.append(".VALUE ");
select.append(tempAlias);
    }
}
select.append(", ");
select.append(gSelect.toString());
}
else {
    select.append(" ");
    if (isReleaseItemTypeSet) select.append(" DISTINCT ");
    select.append(re.getDriverAlias());
    select.append(".");
    select.append(DRIVER_KEY); // we only ever get back list of problem
ids
    if (addReleasesSql) select.append(sortSelect.toString());

    // if (this.hr) {
    //     select.append(" ");
    //     select.append(driverAlias);
    //     select.append(".");
    //     select.append(getHistKey(driverTable, false));
    // }
}

// Build the query string
//-----

// First process the not exists clauses / criteria for each child table
//-----
String tempKey = "";
Iterator necIterator = notExistClauses.keySet().iterator();
ArrayList nullParams = new ArrayList();
StringBuffer andNotExists = new StringBuffer();
String ddeTableColumn = "";
ArrayList tempList = null;
andExists = new StringBuffer();
andOther = new StringBuffer();
paramsOther = new ArrayList();
boolean doOrExists = false;

while (necIterator.hasNext()){
    tempKey = (String)necIterator.next();
    andExists = (StringBuffer) existClauses.get(tempKey);
    andNotExists = (StringBuffer) notExistClauses.get(tempKey);
    if (andExists != null){
        doOrExists = orExists.get(tempKey) != null;

        if (andNotExists != null){
            andOther.append((String)existAndOrs.get(tempKey));
            if (doOrExists) andOther.append(" ( ");

            andOther.append(andNotExists);

```


Report.java

```

        if (doOrExists){
            andOther.append(" or ");
            andOther.append( andExists );
            andOther.append(" ) ");
        }
    }
    if ( ( tempList = (ArrayList)existParams.get(tempKey) ) != null
) paramsOther.addAll(tempList);
    if ( ( tempList = (ArrayList)existTypes.get(tempKey)) != null)
typesOther.addAll(tempList);
    existClauses.remove(tempKey); // this wont get processed again
as an exist
    }
    else if (andNotExists != null){
        andOther.append((String)existAndOrs.get(tempKey));
        andOther.append(andNotExists);
    }
}

// Then process the exists clauses / criteria for each
// child table (but leave out the is null ones)
//-----
Iterator ecIterator = existClauses.keySet().iterator();
while (ecIterator.hasNext()){
    tempKey = (String)ecIterator.next();
    andExists = (StringBuffer) existClauses.get(tempKey);
    if (andExists != null){
        andOther.append((String)existAndOrs.get(tempKey));
        andOther.append( andExists );
        if ( ( tempList = (ArrayList)existParams.get(tempKey) ) != null
) paramsOther.addAll(tempList);
        if ( ( tempList = (ArrayList)existTypes.get(tempKey)) != null)
typesOther.addAll(tempList);
    }
}

// Build the query String
//-----
// -----DEBUG STUFF-----
Z.log.writeToLog(Z.log.DEBUG5,"ABOUT TO buildQueryString " );
Z.log.writeToLog(Z.log.DEBUG5,"select.toString() "+ select.toString() );
Z.log.writeToLog(Z.log.DEBUG5,"from.toString() "+ from.toString());
Z.log.writeToLog(Z.log.DEBUG5,"where.toString() "+ where.toString());
Z.log.writeToLog(Z.log.DEBUG5,"preCondition.toString() "+ preCondition.toString());
Z.log.writeToLog(Z.log.DEBUG5,"andItemType.toString() "+ andItemType.toString());
Z.log.writeToLog(Z.log.DEBUG5,"andProblem.toString() "+ andProblem.toString());
Z.log.writeToLog(Z.log.DEBUG5,"andOther.toString() "+ andOther.toString());
Z.log.writeToLog(Z.log.DEBUG5,"andUDF.toString() "+ andUDF.toString());
Z.log.writeToLog(Z.log.DEBUG5,"andKeyword.toString() "+ andKeyword.toString());
Z.log.writeToLog(Z.log.DEBUG5,"andSortOrder.toString() "+andSortOrder.toString() );
Z.log.writeToLog(Z.log.DEBUG5,"orderBy.toString() "+ orderBy.toString());
Z.log.writeToLog(Z.log.DEBUG5,"groupBy.toString() "+ groupBy.toString());
//-----DEBUG STUFF-----//

    query = buildQueryString(select.toString(),
                            from.toString(),
                            where.toString(),
                            preCondition.toString(),
                            andItemType.toString(),
                            andProblem.toString(),
                            andOther.toString(),
                            andUDF.toString(),

```

```

Report.java
andKeyword.toString(),
andSortOrder.toString(),
/*(isReleaseItemTypeSet) ? " " :*/
orderBy.toString(),
groupBy.toString());

Z.log.writeToLog(Z.log.DEBUG5, "RPT:LIST QUERY IS :" + query);
// if (isReleaseItemTypeSet) query += " ) order by item_id;" ;
statement = conn.prepareStatement(query);

String inSumQuery = "";
String sumAttachSizeQuery = "";

if (doAttachAlert) {
    inSumQuery = buildQueryString(select.toString(),
        from.toString(),
        where.toString(),
        preCondition.toString(),
        andItemType.toString(),
        andProblem.toString(),
        andOther.toString(),
        andUDF.toString(),
        andKeyword.toString(),
        andSortOrder.toString(),
        "");
    sumAttachSizeQuery = "select sum(a.file_size) " +
        " from attachment a where a.item_id in ( " +
        inSumQuery + ")";

    Z.log.writeToLog(Z.log.DEBUG5, "RPT: sumAttachSizeQuery = " +
sumAttachSizeQuery);
    sumAttachStatement = conn.prepareStatement(sumAttachSizeQuery);
}
else if (isNewPersist && !this.isSummaryReport()) {
    countQuery = buildQueryString("select " + iHint + " count( DISTINCT
"+ re.getDriverAlias() + ".ITEM_ID ) ",
        from.toString(),
        where.toString(),
        preCondition.toString(),
        andItemType.toString(),
        andProblem.toString(),
        andOther.toString(),
        andUDF.toString(),
        andKeyword.toString(),
        andSortOrder.toString(),
        "");
    countStatement = conn.prepareStatement(countQuery);
    Z.log.writeToLog(Z.log.DEBUG5, "RPT: COUNT QUERY IS :" +
countQuery);
}

java.sql.Timestamp tempTimestamp = null;

params.addAll(paramsSortOrder);
params.addAll(paramsPreCondition);
int timeSeriesDateParamPositionOffset = params.size();

```

```

                                Report.java
params.addAll(paramsItemType);
params.addAll(paramsProblem);
params.addAll(paramsOther);
params.addAll(paramsUDF);
params.addAll(paramsKeyword);

types.addAll(typesSortOrder);
types.addAll(typesPreCondition);
types.addAll(typesItemType);
types.addAll(typesProblem);
types.addAll(typesOther);
types.addAll(typesUDF);
types.addAll(typesKeyword);

if (isTimeSeries){
    int timeSeriesDayCnt = 1;
    Iterator it = null;
    Integer tmpInt = null;

    ArrayList slices = DbTime.getTimestampsBetweenStartStopLimits(
dbtForTimeSeries.getDbThenTimestampLessOneMinuete(1),
    dbt2.getDbThenTimestampLessOneMinuete(1),
    this.getTimeInterval());

    Timestamp currentSlice = dbt.getDbNowTimestamp();
    for (int w =0; w < (slices.size()); w++){
        currentSlice = (Timestamp)slices.get(w);
        //Z.probe("CURRENT SLICE = " + currentSlice);

        // Combine parameters
        // -----

        // Replace the param values for the timeSeries Dates
        //-----
        for (int j = 0; j < timeSeriesDateParamPositions.length; j++)
            params.set(timeSeriesDateParamPositionOffset +
timeSeriesDateParamPositions[j],
                currentSlice);

        // Bind the params
        //-----
        String bindType = "";

        Z.log.writeToLog(Z.log.DEBUG, "RPT: params : " + params);
        Z.log.writeToLog(Z.log.DEBUG, "RPT: types : " + types);

        statement = this.bindParams(params, types, statement);

        if (doAttachAlert)
            sumAttachStatement = this.bindParams(params,
types,sumAttachStatement);
        else if (isNewPersist && !this.isSummaryReport())
            countStatement = this.bindParams(params,
types,countStatement);

        // Get the result set
        //-----
        ResultSet result = statement.executeQuery();

```

Report.java

```
// SUMMARY REPORT RESULT SET
//-----
if (this.isSummaryReport()) {

    while (result.next()) {
        rows = new HashMap();

/* Retrieve Data for each sort field
-----*/
//Z.probe("soFields.size == " + soFields.size());
//Z.probe("groupByAliases.size=" + groupByAliases.size());
//Z.probe("soFields== " + soFields);
//Z.probe("groupByAliases=" + groupByAliases);

        Iterator gbi = groupByAliases.keySet().iterator();
        int k = 0;

        rows.put("TIMESTAMP",this.formatDateString(dbt,
            su,
            Convert.toCalendar(currentSlice),
            emptyStringSpacer));

        while (gbi.hasNext()) {
            alias = (String) gbi.next();

/* Deal with dates
-----*/

            if (ddeDisplayedDates.get(alias) != null) {
//temp = result.getString(alias);
                tempTimestamp = result.getTimestamp(alias);
                if (TextManager.isStringVisible(temp)) {
                    temp = formatDateString(dbt,
                        su,

//Convert.getCalendarFromString(session, temp),
                        Convert.toCalendar(tempTimestamp),
                        emptyStringSpacer);
                } else temp =
this.getPrintableResult(result.getString(alias));
                rows.put(alias,
                    TextManager.isStringInvisible(temp)
                    ? "CALC".equals(groupByAliases.get(alias))
                    ? "0" : emptyStringSpacer
                    : temp);

/* Deal with values & dates
-----*/
                if ("D".equals(groupByAliases.get(alias))) {
                    tempTimestamp = result.getTimestamp(alias +
valuesSufx); // this was getString()
// Convert.toCalendar(tempTimestamp);
                    if (TextManager.isStringVisible(temp)) {
                        temp = formatDateString(dbt,
                            su,
                            Convert.toCalendar(tempTimestamp),
```

```

Report.java
//Convert.getCalendarFromString(session, result.getString(alias + valueSufx)),
//emptyStringSpacer);
rows.put(alias + valueSufx, temp);
}
}
else if ("Y".equals(groupByAliases.get(alias))) {
temp = result.getString(alias + valueSufx);
rows.put(alias + valueSufx, temp);
}
}
if (this.calculatedFields.size() == 0)
rows.put(idCountAlias,
result.getString(idCountAlias)); // the count

results.add(rows);
} // end while
} // end if summary report
} // end while
} // end if timeseries
else{ // not a timeseries

/* Bind the params
-----*/
String bindType = "";

Z.log.writeToLog(Z.log.DEBUG, "RPT: params : " + params);
Z.log.writeToLog(Z.log.DEBUG, "RPT: types : " + types);

statement = this.bindParams(params, types, statement);
if (doAttachAlert)
sumAttachStatement = this.bindParams(params,
types,sumAttachStatement);
else if (isNewPersist && !this.isSummaryReport())
countStatement = this.bindParams(params, types,countStatement);

/* Get the result set
-----*/
ResultSet result = statement.executeQuery();

int rsRownum = 0;

// currently the largest blocksize is 5000 = 10 pgs * 500 results per page
int lastBlockSize = 0;
int currentBlockSize = 0;

// limited results per page
if (!unlimitedResults) {
if (this.pageBlock > 1 && retrieveCount > 0)
lastBlockSize = (this.pageBlock - 1) * retrieveCount;

// ensure never to have two negatives multiplied to make a positive
currentBlockSize = (this.pageBlock < 0 ? 0 : this.pageBlock) *
(retrieveCount < 0 ? 0 : retrieveCount);
}

// GET THE RESULTS
//-----
// Summary report result set
//-----
if (this.isSummaryReport()) {

```

```

Report.java
while (result.next()) {

    rows = new HashMap();

/* Retrieve Data for each sort field
-----*/
//Z.probe("soFields.size == " + soFields.size());
//Z.probe("groupByAliases.size=" + groupByAliases.size());
//Z.probe("soFields== " + soFields);
//Z.probe("groupByAliases=" + groupByAliases);

    Iterator gbi = groupByAliases.keySet().iterator();
    int k = 0;

    while (gbi.hasNext()) {
        alias = (String) gbi.next();

/* Deal with dates
-----*/

        if (ddeDisplayedDates.get(alias) != null) {
//temp = result.getString(alias);
            tempTimestamp = result.getTimestamp(alias);
            if (TextManager.isStringVisible(temp)) {
                temp = formatDateString(dbt,
                    su,
                    //Convert.getCalendarFromString(session,
temp),
                    Convert.toCalendar(tempTimestamp),
                    emptyStringSpacer);
            } else temp =
this.getPrintableResult(result.getString(alias));

            rows.put(alias,
                TextManager.isStringInvisible(temp)
                ? "CALC".equals(groupByAliases.get(alias))
                ? "0" : emptyStringSpacer
                : temp);

/* Deal with values & dates
-----*/
            if ("D".equals(groupByAliases.get(alias))) {
                tempTimestamp = result.getTimestamp(alias +
valuesuffix); // this was getString()
// Convert.toCalendar(tempTimestamp);
                if (TextManager.isStringVisible(temp)) {
                    temp = formatDateString(dbt,
                        su,
                        Convert.toCalendar(tempTimestamp),
                        //Convert.getCalendarFromString(session,
result.getString(alias + valuesuffix)),
                        emptyStringSpacer);
                    rows.put(alias + valuesuffix, temp);
                }
            } else if ("Y".equals(groupByAliases.get(alias))) {
                temp = result.getString(alias + valuesuffix);
                rows.put(alias + valuesuffix, temp);
            }
        }

        if (this.calculatedFields.size() == 0)

```

```

Report.java
rows.put(idCountAlias, result.getString(idCountAlias));
// the count

        results.add(rows);
    } // end while
} // end if summary report

// NOT A SUMMARY REPORT RESULT SET
//-----
    else if (this.unlimitedResults) {

// LIMIT_QUERY_ROWS is the maximum allowable returned results for an unlimited query
if you are NOT IN the ADMIN_BYPASS_GROUP
        this.checkMaxRows =
!session.getUserRole().equalsIgnoreCase(Z.appDefaults.getAttribute("ADMIN_BYPASS_GRO
UP"));
        if (checkMaxRows) {
            if
(TextManager.isStringVisible(Z.appDefaults.getAttribute("LIMIT_QUERY_ROWS"))) {
                if
(Z.appDefaults.getAttribute("LIMIT_QUERY_ROWS").indexOf("DEFAULT") == -1)
                    this.maxRows =
Integer.parseInt(Z.appDefaults.getAttribute("LIMIT_QUERY_ROWS"));
            }
            Z.log.writeToLog(Z.log.DEBUG1, "RPT: unlimited/checkmax = " +
this.unlimitedResults + "/" + checkMaxRows);
            while (result.next()) {

// if the results are unlimited then limit to 5000 ids in memory at a time
//-----
                rsRownum++;
                flushToPersist(persist, result.getString(1));
                if (this.checkMaxRows) {
                    if (this.maxRows <= rsRownum) break;
                }
            } // end while
        } // end if

// The results are limited to one pageBlock
// where pageBlock = max(25,100,500) x 10 pgs = total ids in memory at a time
//-----
        else{
            else {
                while (result.next()) {
                    if (++rsRownum > (lastBlockSize))
                        flushToPersist(persist, result.getString(1));
                    else continue;
                    if (rsRownum == (currentBlockSize)) break;
                }
            }
            boolean allFetched = result.isAfterLast();

            if (!this.isSummaryReport()) {
                if (doAttachAlert) {
                    ResultSet sumAttachResult =
sumAttachStatement.executeQuery();

```



```

Report.java
        if (sumAttachResult.next()) this.totalAttachmentSize =
sumAttachResult.getInt(1);
        Z.log.writeToLog(Log.DEBUG3, this, "RPT***** THE
THIS.TOTAL_ATTACHMENT_SIZE:" + this.resultCount);

        } // Only do the count if necessary
        else if (!allFetched && countStatement != null) {
            ResultSet countResult = countStatement.executeQuery();

            if (countResult.next()) this.resultCount =
countResult.getInt(1);
            } else if (allFetched) this.resultCount = rsRownum;
        Z.log.writeToLog(Log.DEBUG3, this, "RPT***** THE
THIS.RESULT_COUNT:" + this.resultCount);
    }
    } // end if not a timeSeries

}
catch (Exception e) {
    Z.log.writeToLog(Log.ERROR, this, "Error in method getList:" + e);
    ErrorWriter.write(e, ErrorWriter.LOG);
    throw e;
}
finally {
    try {
        if (statement != null) statement.close();
    }
    catch (Exception e1) {
        Z.log.writeToLog(Log.ERROR, this, "Error in method getList:" + e1);
        ErrorWriter.write(e1, ErrorWriter.LOG);
    }
    try {
        if (countStatement != null) countStatement.close();
    }
    catch (Exception e2) {
        Z.log.writeToLog(Log.ERROR, this, "Error in method getList:" + e2);
        ErrorWriter.write(e2, ErrorWriter.LOG);
    }
    statement = null;
    countStatement = null;
    return results; // if not a summary report then this is 0
}
} //end get masterlist

/**
 * flushToPersist()
 * @param Persist persist
 * @param String results
 */
private void flushToPersist(Persist persist, String result) throws Exception {
    if (isNewPersist) {
        persist.writeNew(result);
        isNewPersist = false;
    } else {
        persist.write(result);
    }
}

private PreparedStatement bindParams(ArrayList params,

```

Report.java

```

        ArrayList types,
        PreparedStatement statement) throws Exception{
    try{
        String bindType = "";
        for (int k = 0; k < params.size(); k++) {
            bindType = (String) types.get(k);
            if (bindType.equalsIgnoreCase("STRING")) {
                statement.setString(k + 1, (String) params.get(k));
            }
            else if (bindType.equalsIgnoreCase("DATE")) {
                statement.setTimestamp(k + 1, (Timestamp) params.get(k));
            }
        }
    }
    catch(Exception e){
        throw e;
    }
    finally{
        return statement;
    }
}

```

```

/*****
****

```

```

****
****

```

```

    *   getSelect()
    *   @return select string
    *
    *   Builds the select clause for an individual record --- does this in blocks
of 20 records at a time.
    *
    *   This is done in three queries due to multiple repeating values associated
    *   with problem modules and problem releases. The three queries are:
    *
    *       1) problem table as driver table
    *       2) problem release as driver table
    *       3) problem module as driver table
    *
    *   Get the data for a block of 20 (this is the smallest page size) using an IN
clause,
    *   the concatenation hint and correlated sub-queries in the select clause
(w whenever possible)
    *   to achieve the FASTEST performance.
    *   (To use correlated sub-queries, you will need to make any field deep in the
    *   table hierarchy a lookup in the data dictionary :
    *   -- ie. module name is a lookup of module.title done from the
problem_module table
    *   -- ie. module type is a lookup of module_type.title done from the module
table )
    *
    *   Then you must SORT the results. To do this you will parse the block of 20
results
    *   to match order the master list, sortedKeys (retrieved from the persist
object)
    *   that was used to generate the IN clause. Therefore you combine the problem
results,
    *   problem release results and problem module results, sequentially by
sortedKey.

```

Report.java

```

*
* So you have processed your block of 20 but the user wants 50 results per
page... First, you
* return the results as a ValidationList (a hashmap that keeps its order FIFO).
This list This means get
* Select gets called again from getNextBlockResults()
*

```

```

*****
*****

```

```

*****
*****/

```

```

private ValidationList getSelect(Connection conn, ArrayList sortedKeys,
SesameSession session) throws Exception {

```

```

/* DDEntries

```

```

-----*/
String ddeType = "";
String ddeTable = "";
String driverKey = "ITEM_ID";
// String problemKey = "ITEM_ID";
String problemUdfDriverKey = "ITEM_ID";
String driverTableAlias = "p";
String tmpDriverTable = "";
String ddeField = "";
String ddeName = "";
String ddeLookupTable = "";
String ddeLookupColumn1 = "";
String ddeLookupColumn2 = "";
String ddeLookupColumn3 = "";
String ddeLookupKey = "";
String ddeParentTable = "";
String ddeGrandParentTable = "";
String ddeGrandParentKey = "";
String ddeGrandChildKey = "";
String ddeParentKey = "";
String ddeChildKey = "";
String ddeDisplayType = "";
String ddeMultipleValue = "";
String ddeItemId = "";
HashMap ddeDisplayedDates = new HashMap();
HashMap prDisplayedDates = new HashMap();
HashMap ddeTextFields = new HashMap();
HashMap ddeDisplayURLs = new HashMap();

```

```

/* Used to build query string

```

```

-----*/
boolean hasGrandParentTable = false;
boolean doPQuery = false;
boolean doPMQuery = false;
boolean doPRQuery = false;
boolean doPAQuery = false;
boolean doRGQuery = false;
boolean doUDFTextQuery = false;
boolean doReleaseUdfTextQuery = false;
boolean doUDFMultiQuery = false;
boolean doReleaseUdfMultiQuery = false;
boolean hasDriverTable = false;
String ORDER_BY = " order by ";
String temp = "";
String aliasedDriverTable = "";
String pkey = ""; // this is the problem Id as a key

```

```

String lookupSql = "";
String key = ""; // this is a generic variable used as a key in HashMaps
StringBuffer inCriteria = new StringBuffer();
JoinData p_jd = new JoinData();
JoinData pr_jd = new JoinData();
JoinData pm_jd = new JoinData();
JoinData pa_jd = new JoinData();

/* Problem Table Query
-----*/
boolean pFromTables = false; // are there tables in the from clause?
int pSelectCnt = 0;
String tempAlias = "";
String pQuery = "";
StringBuffer pSelect = new StringBuffer("select " + this.concatHint);
StringBuffer pFrom = new StringBuffer(" from ");
StringBuffer pwhere = new StringBuffer(" where p." + driverKey + " in ");
ArrayList pParams = new ArrayList(); // Bind Parameters for SQL UDF
fields
ArrayList problemFieldList = new ArrayList(); // the list of ddnames that
are in the problem query
HashMap pRows = null;
HashMap pResults = null;

/* Problem Release Table Query
-----*/
boolean prFromTables = false; // are there tables in the from clause?
int prSelectCnt = 0;
String prQuery = "";
StringBuffer prSelect = new StringBuffer("select " + this.concatHint);
StringBuffer prFrom = new StringBuffer(" from ");
StringBuffer prwhere = new StringBuffer(" where ir.problem_release_id in ");
ArrayList prParams = new ArrayList(); // Bind Parameters for SQL UDF
fields
HashMap prResults = null;
String rQuery = "select item1_id from item_group where item_group_type_id =1
and item2_id in";
StringBuffer rInCriteria = new StringBuffer();
ArrayList releaseFieldList = new ArrayList(); // the list of ddnames that
are in the release query

/* Problem Module Table Query
-----*/
boolean pmFromTables = false; // are there tables in the from clause?
int pmSelectCnt = 0;
String pmQuery = "";
StringBuffer pmSelect = new StringBuffer("select " + this.concatHint);
StringBuffer pmFrom = new StringBuffer(" from ");
// *****CHANGED THIS TO LOOKUP ON ITEM_ID
StringBuffer pmWhere = new StringBuffer(" where im.item_module_id in ");
HashMap pmResults = null;
String mQuery = " select item_module_id " +
" from item_module " +
" where item_module.item_id in ";
StringBuffer mInCriteria = new StringBuffer();
ArrayList moduleFieldList = new ArrayList(); // the list of ddnames that
are in the module query

/* Attachment Table Query
-----*/
boolean paFromTables = false; // are there tables in the from clause?
int paSelectCnt = 0;
String paQuery = "";

```

```

                                Report.java
StringBuffer paSelect = new StringBuffer("select " + this.concatHint);
StringBuffer paFrom = new StringBuffer(" from ");
StringBuffer paWhere = new StringBuffer(" where a.attachment_id in ");
HashMap paResults = null;
String aQuery = " select attachment_id " +
                " from attachment " +
                " where attachment.item_id in ";
StringBuffer aInCriteria = new StringBuffer();
ArrayList attachmentFieldList = new ArrayList(); // the list of ddnames
that are in the attachment

/* Relationship Group Table Query
-----*/
boolean rgFromTables = false; // are there tables in the from clause?
int rgSelectCnt = 0;
String rgQuery = "";
StringBuffer rgSelect = new StringBuffer("select " + this.concatHint);
StringBuffer rgFrom = new StringBuffer(" from ");
StringBuffer rgWhere = new StringBuffer(" where rg.relationship_group_id in
");
HashMap rgResults = null;
String relationshipQuery = " select relationship_group_id " +
                          " from relationship_group_view " +
                          " where relationship_group_view.item_id in ";
StringBuffer rgInCriteria = new StringBuffer();
ArrayList rgFieldList = new ArrayList(); // the list of ddnames that are in
the attachment

/* UDF MultiRow Queries
-----*/
String udfQuery = "";
ArrayList udfNameParams = new ArrayList();
ArrayList udfTextFields = new ArrayList();
ArrayList releaseUdfTextFields = new ArrayList();
ArrayList udfFieldResults = new ArrayList();
HashMap udfResults = null;
final String problemUdfTextKey = "PROBLEM_TEXT_ID";

ArrayList udfMultiFields = new ArrayList();
ArrayList releaseUdfMultiFields = new ArrayList();

/* Data Alias Names used for retrieving results from a result set
-----*/
HashMap pDataAlias = new HashMap();
HashMap pmDataAlias = new HashMap();
HashMap prDataAlias = new HashMap();
HashMap paDataAlias = new HashMap();
HashMap rgDataAlias = new HashMap();

/* For Transacting Queries
-----*/
ResultSet pRs = null;
ResultSet prRs = null;
ResultSet pmRs = null;
ResultSet paRs = null;
ResultSet rgRs = null;
ResultSet udfRs = null;
PreparedStatement pStmt = null;
PreparedStatement prStmt = null;
PreparedStatement pmStmt = null;
PreparedStatement paStmt = null;
PreparedStatement rgStmt = null;
PreparedStatement udfStmt = null;

```

Report.java

```

/* Other
-----*/
String listKey = "";
ArrayList leList = new ArrayList();
ArrayList dataList = null;

ArrayList releaseKeys = null;
ArrayList moduleKeys = null;
ArrayList attachmentKeys = null;
ArrayList rgKeys = null;

HashMap distinctTables = new HashMap();
HashMap tempHash = null;
LayoutElement le = null;
DDEntry thisDDE = null;
ValidationList finalResults = new ValidationList();
HashMap dictionaryFields = new HashMap();
HashMap multiText = new HashMap();
DbTime dbt = null;
SecurityUser su = null;

//          Regex replaceConcat = new Regex(esc+"~|~|",Z.dbms.cat());
//Z.probe("RPT:~~~~~IN GET SELECT~~~~~");

try {

    dbt = new DbTime(session, conn);
    su = (SecurityUser) session.getAttribute("USER");

    /**
     * Make sure the layouts & layout elements are populated
     */

    if (this.l == null) {
        Z.log.writeToLog(Z.log.DEBUG3, "THE LAYOUT IS NULL!!!!!!!!!!!!!!");
        throw new Exception("The layout is null");
    }

    leList = l.getElements();
    if (leList == null) {
        Z.log.writeToLog(Z.log.DEBUG3, "Layout has 0 layout elements and
therefore lei is null");
        throw new Exception("The layout has no elements!!");
    }
    multiLookups = new HashMap();
    dictionaryFields = Z.dictionary.getHashMap();

/* Build the IN clause for getting a blocks worth of data
-----*/
    inCriteria = buildInClause(sortedKeys);

    /**
     * Scan the Layout Elements for ATTACHMENT
     */

    boolean hasAttachment = false;
    boolean hasRG = false;
    String leName = "";
    ArrayList leNameList = new ArrayList();

    for (int i = 0; i < leList.size(); i++) {
        leName = ((LayoutElement) leList.get(i)).getDdName();

```

```

Report.java
Z.log.writeToLog(Z.log.DEBUG3,"THE REPORT LAYOUT ELEMENTS ARE : " +
leName);
    if ("ATTACHMENT".equalsIgnoreCase(leName))
        hasAttachment = true;
    else if ("RELATIONSHIP_GROUP".equalsIgnoreCase(leName))
        hasRG = true;
    else
        leNameList.add(leName);
}
// // add the release one.
// leNameList.add("PROBLEM_RELEASE_ID");

    if (hasAttachment) {
        leNameList.add("ATTACHMENT_ID");
        leNameList.add("ATTACH_FILE_DESC");
        leNameList.add("ATTACH_FILE_NAME");
        leNameList.add("ATTACH_CONTENT_TYPE");
        leNameList.add("ATTACH_FILE_SIZE");
        leNameList.add("ATTACH_DATE_CREATED");
        leNameList.add("ATTACH_PATH");
        leNameList.add("ATTACH_CREATED_BY_USER");
    }

    if (hasRG) {
        leNameList.add("RELATIONSHIP_GROUP_ID");
        leNameList.add("RELATIONSHIP_GROUP_TYPE");
        leNameList.add("RELATIONSHIP_GROUP_TITLE");
        leNameList.add("RELATIONSHIP_GROUP_OWNER");
        leNameList.add("RELATIONSHIP_GROUP_ID");
        leNameList.add("RELATIONSHIP_GRP_PARENT_ID");
    }
    ReportElement re = new ReportElement(null,null,this.isChart(),hr);

    problemFieldList.add(PROBLEM_KEY_ALIAS);
    releaseFieldList.add(PROBLEM_KEY_ALIAS);
    releaseFieldList.add(RELEASE_KEY_ALIAS);
    moduleFieldList.add(PROBLEM_KEY_ALIAS);
    attachmentFieldList.add(PROBLEM_KEY_ALIAS);

    /**
     * Process the Layout Elements
     */
    String baseName = "";
    for (int i = 0; i < leNameList.size(); i++) {
/* Get the Data Dictionary Entry
----- */
        key = (String) leNameList.get(i);

        if ( key.indexOf("RANGE_START") > -1)
            baseName = key.substring(12);
        else if (key.indexOf("RANGE_STOP") > -1)
            baseName = key.substring(11);
        else
            baseName = key;

// only process valid filter fields
//-----
        if ((thisDDE = (DDEntry) dictionaryFields.get(baseName)) == null)
continue;

        thisDDE.setName(baseName);

```



```

Report.java
//      f = this.fg.getFilter(key);

/* Populate the string variables
----- */
      ddeType = thisDDE.getType();
      ddeName = thisDDE.getName();
      ddeTable = thisDDE.getTableName() == null ? "" :
thisDDE.getTableName();

      if (!"PROBLEM_RELEASE".equalsIgnoreCase(ddeTable)){
          if (thisDDE.getItemTypeId() != null)
              ddeItemTypeId = thisDDE.getItemTypeId();
          else ddeItemTypeId = "";
      }

// special case for release parents
//-----
      else{
          ddeItemTypeId = "1";
      }

/* If the Layout Element is not a database object we cannot use it in a query
-----*/
      if (!thisDDE.isSelectForReports()
          && !"ATTACHMENT".equalsIgnoreCase(ddeName)
          && !"RELATIONSHIP_GROUP".equalsIgnoreCase(ddeName)) {

          Z.log.writeToLog(Z.log.DEBUG5,"RPT:___NOT Processing element: "
+ ddeName + "___NOT SELECTABLE FOR REPORTS");
          continue;
      }
      Z.log.writeToLog(Z.log.DEBUG5,"RPT:Processing report results layout
element: " + ddeName + "_____ is selectable FOR REPORTS");

      if (thisDDE.getColumnName() != null)
          ddeField = TextManager.replace(thisDDE.getColumnName(), "||",
Z.dbms.cat());
      else ddeField = "";

      if (thisDDE.getParentTable() != null)
          ddeParentTable = thisDDE.getParentTable();
      else ddeParentTable = "";

      ddeGrandParentTable = thisDDE.getGrandParentTable();
      if (!"".equals(ddeGrandParentTable)) {
          hasGrandParentTable = true;
          ddeGrandParentKey = thisDDE.getGrandParentKey();
          ddeGrandChildKey = thisDDE.getGrandChildKey();
      } else hasGrandParentTable = false;

//Z.probe("AAAAAAAAAAAAAAAAhasGrandParentTableAAAAAAAA:" + hasGrandParentTable);
      if (thisDDE.getParentKey() != null)
          ddeParentKey = TextManager.replace(thisDDE.getParentKey(), "||",
Z.dbms.cat());
      else ddeParentKey = "";

      if (thisDDE.getChildKey() != null)
          ddeChildKey = TextManager.replace(thisDDE.getChildKey(), "||",
Z.dbms.cat());
      else ddeChildKey = "";

```

```

Report.java
    ddeLookupTable = thisSDE.getLookupTable() == null ? "" :
thisSDE.getLookupTable();
    ddeDisplayType = thisSDE.getDisplayType();

    if (thisSDE.getLookupColumn1() != null)
        ddeLookupColumn1 =
TextManager.replace(thisSDE.getLookupColumn1(), "||", Z.dbms.cat());
    else ddeLookupColumn1 = "";

    if (thisSDE.getLookupColumn2() != null)
        ddeLookupColumn2 =
TextManager.replace(thisSDE.getLookupColumn2(), "||", Z.dbms.cat());
    else ddeLookupColumn2 = "";

    if (thisSDE.getLookupColumn3() != null)
        ddeLookupColumn3 =
TextManager.replace(thisSDE.getLookupColumn3(), "||", Z.dbms.cat());
    else ddeLookupColumn3 = "";

    if (thisSDE.getLookupKey() != null)
        ddeLookupKey = TextManager.replace(thisSDE.getLookupKey(), "||",
Z.dbms.cat());
    else ddeLookupKey = "";

    ddeMultipleValue = thisSDE.isMultipleValue() ? "Y" : "N";

    if ("IMAGE".equalsIgnoreCase(ddeDisplayType)) continue; // skip
images
    if (thisSDE.displayAsURL()) ddeDisplayURLs.put(ddeName,
thisSDE.getUrl());
//Z.probe("RPT:ddeDisplayURLs: " + ddeDisplayURLs);
//Z.probe("RPT:Processing layout element: " + ddeName + "      IS IT MULTI: " +
thisSDE.isMultipleValue());

```

```

//*****
//
// SET THE DRIVER TABLES
//
//*****

```

```

    hasDriverTable = false;

    if (("ITEM".equalsIgnoreCase(ddeTable)
        || "ITEM".equalsIgnoreCase(ddeParentTable)
        || "PROBLEM".equalsIgnoreCase(ddeTable)
        || "PROBLEM".equalsIgnoreCase(ddeParentTable)
        || "RELATIONSHIP_GROUP".equalsIgnoreCase(ddeTable)
        || "RELATIONSHIP_GROUP_PROBLEM".equalsIgnoreCase(ddeParentTable)
        || ddeTable.startsWith("RELATIONSHIP_GROUP"))
        && !("PROBLEM_MODULE".equalsIgnoreCase(ddeTable)
            || "PROBLEM_RELEASE".equalsIgnoreCase(ddeTable)
            || "ATTACHMENT".equalsIgnoreCase(ddeTable)
            || (ddeTable.startsWith("ITEM_MODULE")
                || (ddeTable.startsWith("ITEM_RELEASE"))
            ))
    )
    {
        tmpDriverTable = this.driverTable;
        driverKey = "ITEM_ID";
        hasDriverTable = true;
    }

```

```

Report.java
        if (! PROBLEM_KEY_ALIAS.equalsIgnoreCase(ddeName))
problemFieldList.add(ddeName);
    }
    else if ("ITEM_MODULE".equalsIgnoreCase(ddeTable)
        || "ITEM_MODULE".equalsIgnoreCase(ddeParentTable)
        || "PROBLEM_MODULE".equalsIgnoreCase(ddeTable)
        || "PROBLEM_MODULE".equalsIgnoreCase(ddeParentTable)
        || (ddeTable).startsWith("ITEM_MODULE")) {
        tmpDriverTable = "ITEM_MODULE";
        driverKey = "ITEM_MODULE_ID";
        hasDriverTable = true;
        if (! PROBLEM_KEY_ALIAS.equalsIgnoreCase(ddeName))
moduleFieldList.add(ddeName);
    }
    else if ("PROBLEM_RELEASE".equalsIgnoreCase(ddeTable)
        || "PROBLEM_RELEASE".equalsIgnoreCase(ddeParentTable)
        || (ddeTable).startsWith("ITEM_RELEASE")) {
        tmpDriverTable = "PROBLEM_RELEASE";
        driverKey = "PROBLEM_RELEASE_ID";
        hasDriverTable = true;
        if (!PROBLEM_KEY_ALIAS.equalsIgnoreCase(ddeName)
            && !RELEASE_KEY_ALIAS.equalsIgnoreCase(ddeName))
releaseFieldList.add(ddeName);
    }
    else if ("ATTACHMENT".equalsIgnoreCase(ddeTable)) {
        tmpDriverTable = "ATTACHMENT";
        driverKey = "ATTACHMENT_ID";
        hasDriverTable = true;
        if (! PROBLEM_KEY_ALIAS.equalsIgnoreCase(ddeName))
attachmentFieldList.add(ddeName);
    }
    else {
        Z.log.writeToLog(this, Log.DEBUG4, "Couldn't set driver table
because we are too many levels out... DDETABLE =" + ddeTable);
        tmpDriverTable = "";
        hasDriverTable = false;
    }
}

```

```

//*****
//* Process the Query String - UDF Database Objects
//*****

        if ("UDF".equalsIgnoreCase(ddeType)) {

            if ("1".equalsIgnoreCase(ddeItemId)){
                releaseFieldList.add(ddeName);
                tmpDriverTable = "PROBLEM_RELEASE";
                driverKey = "PROBLEM_RELEASE_ID";

                if (!prFromTables){
// Build the IN clause for getting a blocks worth of release stuff
//-----
                    releaseKeys = buildReleaseInList( sortedKeys, rQuery +
inCriteria.toString(), suppressedMultiReleaseFilters);
                    rInCriteria = buildInClause(releaseKeys);
                    Z.probe(" > > > Processing Release Level UDF --
rInCriteria is " + rInCriteria);
                    Z.probe(" > > > Processing Release Level UDF --
releaseKeys is " + releaseKeys);

```

```

Report.java
prWhere.append(rInCriteria.toString());
//prFrom.append(aliasedDriverTable);
pr_jd.addAll(tmpDriverTable, getAlias(tmpDriverTable), "", "", "", "", 0);
prFromTables = true;
distinctTables.put(tmpDriverTable, thisDDE);
}
}
else{
problemFieldList.add(ddeName);
tmpDriverTable = "ITEM";
driverKey = PROBLEM_KEY;

if (!pFromTables) {
prWhere.append(inCriteria.toString());

pr_jd.addAll(tmpDriverTable,
getAlias(tmpDriverTable), "", "", "", "", 0);
//pFrom.append(tmpDriverTable + " " + getAlias(tmpDriverTable));

pFromTables = true;
distinctTables.put(tmpDriverTable, thisDDE);
}
}

Z.log.writeToLog(this, Log.DEBUG4, " >>> Processing UDF : " +
ddeName);

tempAlias = ddeName;
if ("LIST".equalsIgnoreCase(ddeDisplayType)
|| "POPUP".equalsIgnoreCase(ddeDisplayType)
|| "TAB".equalsIgnoreCase(ddeDisplayType)) {
if (thisDDE.isMultipleValueUdf()) {
if ("1".equalsIgnoreCase(ddeItemId)) {
tmpDriverTable = "ITEM_UDF";
releaseUdfMultiFields.add(ddeName);
doReleaseUdfMultiQuery = true;
}
else{
tmpDriverTable = "ITEM_UDF";
udfMultiFields.add(ddeName);
doUDFMultiQuery = true;
}
}
else {
lookupSql = "( select distinct udf_list.title TITLE " +
" from item_udf, udf_list, udf " +
" where item_udf." + PROBLEM_KEY + " = " +
getAlias(tmpDriverTable) + "." + driverKey +
" and udf_list.udf_list_id =
item_udf.udf_list_id " +
" and udf.udf_id = item_udf.udf_id " +
" and udf.name = ? ) " + q + tempAlias + q;

if ("1".equalsIgnoreCase(ddeItemId))
prParams.add(ddeName);
else pParams.add(ddeName);

```

```

    }
    }
    else if ("DATE".equalsIgnoreCase(ddeDisplayType)) {
        lookupSql = " (select distinct item_udf.value_date " +
            " from item_udf, udf " +
            " where item_udf." + PROBLEM_KEY + " = " +
getAlias(tmpDriverTable)+ "." + driverKey +
            " and udf.udf_id = item_udf.udf_id "+
            " and udf.name = ? ) " + q + tempAlias + q;

        if ("1".equalsIgnoreCase(ddeItemId)) {
            prParams.add(ddeName);
            prDisplayedDates.put(ddeName, "");
        }
        else {
            pParams.add(ddeName);
            ddeDisplayedDates.put(ddeName, "");
        }
    }
    else if ("NUMBER".equalsIgnoreCase(ddeDisplayType)) {
        lookupSql = " (select distinct item_udf.value_number " +
            " from item_udf, udf " +
            " where item_udf." + PROBLEM_KEY + " = " +
getAlias(tmpDriverTable)+ "." + driverKey +
            " and udf.udf_id = item_udf.udf_id " +
            " and udf.name = ? ) " + q + tempAlias + q;
        if ("1".equalsIgnoreCase(ddeItemId))
prParams.add(ddeName);
        else pParams.add(ddeName);
    }
    else if ("LOGAREA".equalsIgnoreCase(ddeDisplayType)
        || "TEXTAREA".equalsIgnoreCase(ddeDisplayType)
        || "PRINTTEXT".equalsIgnoreCase(ddeDisplayType)) {

        if ("1".equalsIgnoreCase(ddeItemId)) {
            releaseUdfTextFields.add(ddeName);
            doReleaseUdfTextQuery = true;
        }
        else {
            udfTextFields.add(ddeName);
            doUDFTextQuery = true;
        }
        tmpDriverTable = "ITEM_TEXT";
    }
    else if ("USER".equalsIgnoreCase(ddeDisplayType)) {
        lookupSql =
            " (select distinct security_user.first_name " +
            " from item_udf, udf, security_user " +
            " where item_udf.item_id = "
+getAlias(tmpDriverTable)+ "." + driverKey +
            " and security_user.security_user_id =
item_udf.value" +
            " and udf.udf_id = item_udf.udf_id " +
            " and udf.name = ? ) " + q + tempAlias + q +
            ", (select distinct security_user.last_name " +
            " from item_udf, udf, security_user " +
            " where item_udf.item_id =
"+getAlias(tmpDriverTable)+ "." + driverKey +
            " and security_user.security_user_id =
item_udf.value" +
            " and udf.udf_id = item_udf.udf_id " +
            " and udf.name = ? ) " + q + tempAlias + this.suffix1

```

Report.java

```

+ q +
", (select distinct security_user.email " +
" from item_udf, udf, security_user " +
" where item_udf.item_id =
"+getAlias(tmpDriverTable)+ "." + driverKey +
" and security_user.security_user_id =
item_udf.value" +
" and udf.udf_id = item_udf.udf_id " +
" and udf.name = ? ) " + q + tempAlias + this.sufx2
+ q +
", (select distinct item_udf.value " +
" from item_udf, udf " +
" where item_udf.item_id =
"+getAlias(tmpDriverTable)+ "." + driverKey +
" and udf.udf_id = item_udf.udf_id " +
" and udf.name = ? ) " + q + tempAlias + this.sufx3+
q;

```

```

if ("1".equalsIgnoreCase(ddeItemId)){
    for (int n = 0; n < 4; n++) prParams.add(ddeName);
}
else{
    for (int n = 0; n < 4; n++) pParams.add(ddeName);
}

```

```

}
else{
    lookupSql = " (select distinct item_udf.value " +
" from item_udf, udf " +
" where item_udf." + PROBLEM_KEY + " =
"+getAlias(tmpDriverTable)+ "." + driverKey +
" and udf.udf_id = item_udf.udf_id " +
" and udf.name = ? ) " + q + tempAlias + q;

```

```

prParams.add(ddeName); if ("1".equalsIgnoreCase(ddeItemId))
else pParams.add(ddeName);
}
}

```

```

/*****
* Process the Query String - NON-UDF Database Fields
*****/

```

```

else {

```

```

// Track the dates that will need formatting
//-----
if ("DATE".equalsIgnoreCase(ddeDisplayType))
ddeDisplayedDates.put(ddeName, "");
if ("TEXTFIELD".equalsIgnoreCase(ddeDisplayType))
ddeTextFields.put(ddeName, "");

```

```

/*****
* Process the Query String - Build the from / where clause

```

```

*****/

```

```

Report.java
if (hasDriverTable) {
    if (distinctTables.get(ddeTable) == null) { // get distinct
table
        aliasedDriverTable = tmpDriverTable + " " +
getAlias(tmpDriverTable);

        if (!pFromTables &&
"ITEM".equalsIgnoreCase(tmpDriverTable)) {
            pWhere.append(inCriteria.toString());
            p_jd.addAll(tmpDriverTable,
getAlias(tmpDriverTable), "", "", "", "", 0);
            pFromTables = true;
        }
        else if (!pmFromTables &&
"ITEM_MODULE".equalsIgnoreCase(tmpDriverTable)) {
/* Build the IN clause for getting a blocks worth of module stuff
-----*/
            moduleKeys = buildInList(sortedKeys, mQuery +
inCriteria.toString() /*+
                                ORDER_BY + "ITEM_MODULE_ID DESC"*/);

            mInCriteria = buildInClause(moduleKeys);
            pmWhere.append(mInCriteria.toString());
//pmFrom.append(aliasedDriverTable);
pm_jd.addAll(tmpDriverTable, getAlias(tmpDriverTable), "", "", "", "", 0);
            pmFromTables = true;
        }
        else if (!prFromTables &&
"PROBLEM_RELEASE".equalsIgnoreCase(tmpDriverTable)) {
/* Build the IN clause for getting a blocks worth of release stuff
-----*/
            releaseKeys = buildReleaseInList( sortedKeys, rQuery
+ inCriteria.toString(), suppressedMultiReleaseFilters);
            rInCriteria = buildInClause(releaseKeys);

            prWhere.append(rInCriteria.toString());
//prFrom.append(aliasedDriverTable);
pr_jd.addAll(tmpDriverTable, getAlias(tmpDriverTable), "", "", "", "", 0);
            prFromTables = true;
        }
        else if (!paFromTables &&
"ATTACHMENT".equalsIgnoreCase(tmpDriverTable)) {
/* Build the IN clause for getting a blocks worth of release stuff
-----*/
            attachmentKeys = buildInList(sortedKeys, aQuery +
inCriteria.toString()
                                /* + ORDER_BY + "DATE_CREATED DESC"*/);
            if (attachmentKeys.size() == 0){
                doPAQuery = false;
                continue;
            }
            aInCriteria =
buildInClause(attachmentKeys);
            paWhere.append(aInCriteria.toString());
pa_jd.addAll(tmpDriverTable, getAlias(tmpDriverTable), "", "", "", "", 0);

```



```

Report.java
//paFrom.append(aliasDriverTable);
                                paFromTables = true;
                                }

// If parent table is driver table - one table out
//-----
                                Z.probe("~:o)->-< @@@@ ddeName = " + ddeName );
                                Z.probe("~:o)->-< @@@@ hasGrandParentTable = " +
hasGrandParentTable);
                                Z.probe("~:o)->-< @@@@ tmpDriverTable = " +
tmpDriverTable);
                                Z.probe("~:o)->-< @@@@ ddeParentTable = " +
ddeParentTable);
                                if (ddeParentTable.equalsIgnoreCase(tmpDriverTable) ||
hasGrandParentTable) {

                                if ("ITEM".equalsIgnoreCase(tmpDriverTable)){

                                p_jd.addAll(ddeTable,
                                getAlias(ddeTable),
                                p_jd.LEFT_JOIN,
                                ddeChildKey,
                                hasGrandParentTable ? ddeParentTable :
getAlias(ddeParentTable),
                                ddeParentKey,
                                1);

                                if (hasGrandParentTable &&
distinctTables.get(ddeParentTable) == null){
                                pr_jd.addAll(ddeParentTable,
                                ddeParentTable,
                                p_jd.LEFT_JOIN,
                                ddeGrandChildKey,
                                ddeGrandParentTable,
                                ddeGrandParentKey,
                                2);
                                }
                                }
                                else if
(tmpDriverTable.startsWith("PROBLEM_RELEASE")){

                                if
(ddeTable.equalsIgnoreCase("PRODUCT_RELEASE")){
                                temp = "PRODUCT_RELEASE.PRODUCT_NAME " +
Z.dbms.ojequals() + getAlias(tmpDriverTable) + ".PRODUCT_NAME " +
" and PRODUCT_RELEASE.RELEASE " +
Z.dbms.ojequals() + getAlias(tmpDriverTable) + ".RELEASE_FOUND ";
                                prWhere.append(temp);
                                prFrom.append(ddeTable);
                                }
                                else{
                                pr_jd.addAll(ddeTable,
                                getAlias(ddeTable),
                                p_jd.LEFT_JOIN,
                                ddeChildKey,
                                hasGrandParentTable ? ddeParentTable
: getAlias(ddeParentTable),
                                ddeParentKey,
                                1);
                                }
                                if (hasGrandParentTable &&
distinctTables.get(ddeParentTable) == null){
                                pr_jd.addAll(ddeParentTable,

```

```

Report.java
        ddeParentTable,
        p_jd.LEFT_JOIN,
        ddeGrandChildKey,
        ddeGrandParentTable,
        ddeGrandParentKey,
        2);
    }
}
else if (tmpDriverTable.startsWith("ITEM_MODULE")){
    pm_jd.addAll(ddeTable,
        getAlias(ddeTable),
        p_jd.LEFT_JOIN,
        ddeChildKey,
        hasGrandParentTable ? ddeParentTable :
getAlias(ddeParentTable),
        ddeParentKey,
        1);

    if (hasGrandParentTable &&
distinctTables.get(ddeParentTable) == null){
        pm_jd.addAll(ddeParentTable,
            ddeParentTable,
            p_jd.LEFT_JOIN,
            ddeGrandChildKey,
            ddeGrandParentTable,
            ddeGrandParentKey,
            2);
    }
}

}
distinctTables.put(ddeTable, thisDDE);
} // end build from/ where

/*****
*   BUID THE SELECT CLAUSE
*****/

lookupsql = buildLookupsql(ddeName,
    ddeTable,
    ddeField,
    driverKey,
    ddeChildKey,
    ddeParentTable,
    ddeParentKey,
    ddeGrandParentTable,
    ddeGrandParentKey,
    ddeGrandChildKey,
    ddeLookupTable,
    ddeLookupColumn1,
    ddeLookupColumn2,
    ddeLookupColumn3,
    ddeLookupKey,
    ddeDisplayType,
    ddeMultipleValue);

Z.probe(" ddeName, = "+ ddeName);
Z.probe(" ddeTable, = "+ ddeTable );

```

```

                                Report.java
Z.probe(" ddeField,= "+ ddeField);
Z.probe(" driverKey,= "+ driverKey);
Z.probe(" ddeChildKey,= "+ ddeChildKey );
Z.probe(" ddeParentTable,= "+ ddeParentTable );
Z.probe(" ddeParentKey,= "+ ddeParentKey );
Z.probe(" ddeGrandParentTable,= "+ ddeGrandParentTable );
Z.probe(" ddeGrandParentKey,= "+ ddeGrandParentKey );
Z.probe(" ddeGrandChildKey,= "+ ddeGrandChildKey);
Z.probe(" ddeLookupTable,= "+ ddeLookupTable);
Z.probe(" ddeLookupColumn1,= "+ ddeLookupColumn1 );
Z.probe(" ddeLookupColumn2,= "+ ddeLookupColumn2);
Z.probe(" ddeLookupColumn3,= "+ ddeLookupColumn3 );
Z.probe(" ddeLookupKey,= "+ ddeLookupKey );
Z.probe(" ddeDisplayType,= "+ddeDisplayType );
Z.probe(" ddeMultipleValue= "+ ddeMultipleValue );

Z.probe("@;^)->-<  lookupsql is : " + lookupSql);

} // end if has driver table
} // end if/else UDF

/*****
 * BUID THE SELECT CLAUSE - PROBLEM TABLE
 *****/

if ("ITEM".equalsIgnoreCase(tmpDriverTable)) {

    //Z.probe("RPT: DOING ITEM TABLE QUERY STRING STUFF");
    // Always include the driver table primary key -- ALWAYS
    //-----
    if (!doPQuery) {
        temp = getAlias(tmpDriverTable) + "." + PROBLEM_KEY + " " +
q + PROBLEM_KEY_ALIAS + q + ",";
        pDataAlias.put(PROBLEM_KEY_ALIAS, ddeMultipleValue);
        //Z.probe("RPT: DOING ITEM TABLE QUERY STRING STUFF just set
temp : " +temp);
        doPQuery = true;
    }

    // Build the select clause
    //-----
    if (pSelectCnt++ > 0) pSelect.append(", ");
    else pSelect.append(temp); // must always have this

    pSelect.append(lookupSql);

    // Save the data aliases - parameters are from lookupSql
    //-----
    if ("UDF".equalsIgnoreCase(ddeType)){
        if ("USER".equalsIgnoreCase(ddeType)){
            pDataAlias.put(tempAlias, ddeMultipleValue);
            pDataAlias.put(tempAlias+this.suffix1, ddeMultipleValue);
            pDataAlias.put(tempAlias+this.suffix2, ddeMultipleValue);
            pDataAlias.put(tempAlias+this.suffix3, ddeMultipleValue);
        }
        else
            pDataAlias.put(tempAlias, ddeMultipleValue);
    }
    else {
        Iterator dai = dataAlias.keySet().iterator();

        while (dai.hasNext()) {

```



```

                                Report.java
        if (prSelectCnt++ > 0) prSelect.append(", ");
        else prSelect.append(temp); // must always have this

        prSelect.append(lookupSql);

// Save the data aliases - parameters are from lookupSql
//-----
        if ("UDF".equalsIgnoreCase(ddeType)){
            if ("USER".equalsIgnoreCase(ddeType)){
                prDataAlias.put(tempAlias, thisDDE.isMultipleValueUdf()
? "Y" : "N");
                prDataAlias.put(tempAlias+this.suffix1,
thisDDE.isMultipleValueUdf() ? "Y" : "N");
                prDataAlias.put(tempAlias+this.suffix2,
thisDDE.isMultipleValueUdf() ? "Y" : "N");
                prDataAlias.put(tempAlias+this.suffix3,
thisDDE.isMultipleValueUdf() ? "Y" : "N");
            }
            else
                prDataAlias.put(tempAlias, thisDDE.isMultipleValueUdf()
? "Y" : "N" );
        }
        else{
            Iterator dai = dataAlias.keySet().iterator();

            while (dai.hasNext()) {
                temp = (String) dai.next();
                prDataAlias.put(temp, (String) dataAlias.get(temp));
            }
        }
//    }
} // end building the select clauses

/*****
*   BUILD THE SELECT CLAUSE - ATTACHMENT TABLE
*****/

        else if ("ATTACHMENT".equalsIgnoreCase(tmpDriverTable)) {

/* Always include the driver table primary key -- ALWAYS
-----*/
            if (!doPAQuery) {
                temp = getAlias(tmpDriverTable) + "." + PROBLEM_KEY + " " +
q + PROBLEM_KEY_ALIAS + q + ", ";
                paDataAlias.put(PROBLEM_KEY_ALIAS, ddeMultipleValue);
                doPAQuery = true;
            }

/* Build the select clause
-----*/
            if (paSelectCnt++ > 0) paSelect.append(", ");
            else paSelect.append(temp); // must always have this

            paSelect.append(lookupSql);

/* Save the data aliases - parameters are from lookupSql
-----*/
            Iterator dai = dataAlias.keySet().iterator();

            while (dai.hasNext()) {
                temp = (String) dai.next();
                paDataAlias.put(temp, (String) dataAlias.get(temp));
            }
        }
    }
}

```

```

    }
    } // end if ATTACHEMNT = tmpDriverTable
} //end for loop of layout elements

String[] p_s = Z.dbms.buildFromClause(p_jd);
pFrom.append(p_s[0]);
pwhere.append(p_s[1]);

Z.probe("=====");
Z.probe("@@@@@@@@@@@@@@@@@ QueryBuilder.buildFromClause(jd)[0] P_FROM");
CLAUSE : "+ p_s[0]);
Z.probe("@@@@@@@@@@@@@@@@@ QueryBuilder.buildFromClause(jd)[1] P_WHERE");
CLAUSE : "+ p_s[1]);
Z.probe("-");

String[] pr_s = Z.dbms.buildFromClause(pr_jd);
prFrom.append(pr_s[0]);
prwhere.append(pr_s[1]);

Z.probe("@@@@@@@@@@@@@@@@@ QueryBuilder.buildFromClause(jd)[0] PR_FROM");
CLAUSE : "+ pr_s[0]);
Z.probe("@@@@@@@@@@@@@@@@@ QueryBuilder.buildFromClause(jd)[1] PR_WHERE");
CLAUSE : "+ pr_s[1]);
Z.probe("-");

String[] pm_s = Z.dbms.buildFromClause(pm_jd);
pmFrom.append(pm_s[0]);
pmwhere.append(pm_s[1]);

Z.probe("@@@@@@@@@@@@@@@@@ QueryBuilder.buildFromClause(jd)[0] PM_FROM");
CLAUSE : "+ pm_s[0]);
Z.probe("@@@@@@@@@@@@@@@@@ QueryBuilder.buildFromClause(jd)[1] PM_WHERE");
CLAUSE : "+ pm_s[1]);
Z.probe("-");

String[] pa_s = Z.dbms.buildFromClause(pa_jd);
paFrom.append(pa_s[0]);
pawhere.append(pa_s[1]);

Z.probe("-");
Z.probe("@@@@@@@@@@@@@@@@@ QueryBuilder.buildFromClause(jd)[0] PA_FROM");
CLAUSE : "+ pa_s[0]);
Z.probe("@@@@@@@@@@@@@@@@@ QueryBuilder.buildFromClause(jd)[1] PA_WHERE");
CLAUSE : "+ pa_s[1]);

Z.probe("=====");

/* Build query string from its constituent parts
-----*/
if (doPQuery) pQuery = pSelect.toString() + pFrom.toString() +
pwhere.toString();
if (doPMQuery) pmQuery = pmSelect.toString() + pmFrom.toString() +
pmwhere.toString() + " ORDER BY im.MODULE_ID DESC";
if (doPRQuery) prQuery = prSelect.toString() + prFrom.toString() +
prwhere.toString() + " ORDER BY ir.DATE_CREATED DESC";
if (doPAQuery) paQuery = paSelect.toString() + paFrom.toString() +
pawhere.toString() + " ORDER BY a.DATE_CREATED ASC";

Z.log.writeToLog(Z.log.DEBUG, "RPT:pQuery:" + pQuery);

```

```

Report.java
Z.log.writeToLog(Z.log.DEBUG, "RPT:pmQuery:" + pmQuery);
Z.log.writeToLog(Z.log.DEBUG, "RPT:prQuery:" + prQuery);
Z.log.writeToLog(Z.log.DEBUG, "RPT:paQuery:" + paQuery);

/*****
 * Execute the query -- PROBLEM Driver Table
 *****/

String baseKey = "";
String multi = "";
String multi1 = "";
String multi2 = "";
String multi3 = "";
ArrayList lMulti = null;
ArrayList lMulti1 = null;
ArrayList lMulti2 = null;
ArrayList lMulti3 = null;
DDEntry ddeTemp = null;
int udfCnt = 0;

if (doPQuery) {
    try {
        HashMap pMultiLookups = null;

        pResults = new HashMap();
        pStmt = conn.prepareStatement(pQuery);
        pKey = "";
        udfCnt = pParams.size();

// Bind the UDF Parameters
//-----
        for (int k = 0; k < udfCnt; k++)
            pStmt.setString(k + 1, (String) pParams.get(k));

// Bind the block of problem IDs
//-----
        for (int k = 0; k < sortedKeys.size(); k++)
            pStmt.setString(k + 1 + udfCnt, (String) sortedKeys.get(k));

// Execute the query
//-----
        try {
            pRs = pStmt.executeQuery();
        }
        catch (SQLException se) {
            Z.log.writeToLog(Log.ERROR, this, "Error in method
getSelect:" + se);
            if (pStmt != null) pStmt.close();
            return new ValidationList();
        }
        Set pdaSet = pDataAlias.keySet();

// Process the result rest
//-----
        while (pRs.next()) {
            pRows = new HashMap();
            pMultiLookups = new HashMap();
            Iterator pdaIterator = pdaSet.iterator();

```



```

Report.java
// Get the data comprising a record
// -----
while (pdaIterator.hasNext()) {
    key = (String) (pdaIterator.next());

    if (PROBLEM_KEY_ALIAS.equalsIgnoreCase(key))
        pKey = pRs.getString(key);

// Get the base key (ddName) from the alias
// -----
    if (key.endsWith(sufx1) || key.endsWith(sufx2) ||
key.endsWith(sufx3)) //use sufx1~2~3 for internal processing
        baseKey = key.substring(0, key.length() -
sufx1.length());
    else
        baseKey = key;

    ddeTemp = (DDEntry) dictionaryFields.get(baseKey);
    if ("USER".equalsIgnoreCase(ddeTemp == null ? "" :
ddeTemp.getDisplayType())) {

// Check for a multi lookup --> currently ONLY when display_type = user
// --> TODO: display_as_url = y
// -----
//Z.probe("RPT:BASEKEY ____ ---- : " + baseKey);
//Z.probe("RPT:ddeTemp >>>" +baseKey);

        if (pMultiLookups.get(baseKey) == null) { // only
process unprocessed fields

            multi1 = "";
            multi2 = "";
            multi3 = "";

// LOOKUP1
            multi = pRs.getString(baseKey);
            pMultiLookups.put(baseKey, baseKey);

// LOOKUP2
            multi1 = pRs.getString(baseKey+sufx1);
            pMultiLookups.put(baseKey+sufx1,baseKey);

// LOOKUP3
            multi2 = pRs.getString(baseKey+sufx2);
            pMultiLookups.put(baseKey+sufx2,baseKey);

// LOOKUP4
            multi3 = pRs.getString(baseKey+sufx3);
            pMultiLookups.put(baseKey+sufx3,baseKey);

            pRows.put(baseKey, this.getEmailLink(multi3,
multi,
multi1,
multi2));
        }
    }
    else {
// get the result... format based on diplay type
//-----

```

```

Report.java
    if (ddeDisplayedDates.get(key) != null) {
        temp = formatDateString(dbt,
                                su,
                                Convert.toCalendar(pRs.getTimestamp(key)),
                                emptyStringSpacer);
    }
    else if (ddeDisplayURLs.get(key) != null) {
        temp = getURLLink((String)
ddeDisplayURLs.get(key),
                                pRs.getString(key));
    }
    else if (ddeTextFields.get(key) != null) {
        temp = this.isTextOutputType() ?
pRs.getString(key)
                                :
TextManager.convertUrl(getPrintableResult(pRs.getString(key)));
    }
    else
        temp = getPrintableResult(pRs.getString(key));
    pRows.put(key, temp);
} // end if processed user type
}
pResults.put(pKey, pRows);
}
} catch (Exception e) {
    Z.log.writeToLog(Log.ERROR, "Error in method getSelect:" + e);
//ErrorWriter.write(e, ErrorWriter.LOG);
    throw e;
} finally {
    if (pStmt != null) pStmt.close();
}
}
//Z.probe("RPT:HashMap PROBLEM RESULTS"+pResults);

/*****
*   Execute the query -- PROBLEM_MODULE Driver Table
*****/

    if (doPMQuery) {
//Z.probe("RPT:<><><>moduleKeys = " + moduleKeys);
        try {
            if (moduleKeys == null) moduleKeys = new ArrayList();
            if (moduleKeys.size() == 0)
                pmResults = new HashMap();
            else {
                pmStmt = conn.prepareStatement(pmQuery);

/* Bind the block of problem IDs
-----*/
                for (int k = 0; k < moduleKeys.size(); k++)
                    pmStmt.setString(k + 1, (String) moduleKeys.get(k));

/* Execute the query and process the result set
-----*/
                pmRs = pmStmt.executeQuery();

                pmResults = processResults(pmRs,
                    pmDataAlias,
                    ddeDisplayedDates,

```

```

                                Report.java
                                ddeTextFields,
                                ddeDisplayURLs,
                                dictionaryFields,
                                session);

// Z.probe("RPT:HashMap PROBLEM MODULE RESULTS =" + pmResults);
    }
    } catch (Exception e) {
        Z.log.writeToLog(Log.ERROR, "Error in method getSelect:" + e);
//ErrorWriter.write(e, ErrorWriter.LOG);
        throw e;
    } finally {
        if (pmStmt != null) pmStmt.close();
    }
}

/*****
 * Execute the query -- PROBLEM_RELEASE Driver Table
 *****/

if (doPRQuery) {
    Z.log.writeToLog(Z.log.DEBUG5, "RPT:<><><>prQuery = " + prQuery);
    Z.log.writeToLog(Z.log.DEBUG5, "RPT:<><><>releaseKeys = " +
releaseKeys);
    try {
        if (releaseKeys == null){
            releaseKeys = new ArrayList();
            prResults = new HashMap();
            doPRQuery = false;
        }
        else if (releaseKeys.size() == 0){
            prResults = new HashMap();
            doPRQuery = false;
        }
        else {
            prStmt = conn.prepareStatement(prQuery);

            udfCnt = prParams.size();

// Bind the UDF Parameters
//-----
            for (int k = 0; k < udfCnt; k++)
                prStmt.setString(k + 1, (String) prParams.get(k));

/* Bind the block of problem IDs
-----*/
            for (int k = 0; k < releaseKeys.size(); k++) {
//Z.probe((String)releaseKeys.get(k)+",");
                prStmt.setString(k + 1 + udfCnt, (String)
releaseKeys.get(k));
            }

/* Execute the query and process the result set
-----*/
            prRs = prStmt.executeQuery();
            prResults = processResults(prRs,
                prDataAlias,
                prDisplayedDates,
                ddeTextFields,
                ddeDisplayURLs,
                dictionaryFields,
                session);

```

Report.java

```

        Z.probe("RPT:HashMap PROBLEM RELEASE RESULTS =" + prResults);
    }
}
catch (Exception e) {
    Z.log.writeToLog(Log.ERROR, "Error in method getSelect:" + e);
//ErrorWriter.write(e, ErrorWriter.LOG);
    throw e;
}
finally {
    if (prStmt != null) prStmt.close();
}
}

/*****
 * Execute the query -- ATTACHEMENT Driver Table
 *****/

if (doPAQuery) {
    try {
        if (attachmentKeys == null) attachmentKeys = new ArrayList();
        if (attachmentKeys.size() == 0) paResults = new HashMap();
        else {
            paStmt = conn.prepareStatement(paQuery);

/* Bind the block of problem IDs
-----*/
            for (int k = 0; k < attachmentKeys.size(); k++)
                paStmt.setString(k + 1, (String) attachmentKeys.get(k));

/* Execute the query and process the result set
-----*/
            paRs = paStmt.executeQuery();
            paResults = processResults(paRs,
                paDataAlias,
                ddeDisplayedDates,
                ddeTextFields,
                ddeDisplayURLs,
                dictionaryFields,
                session);

            Z.probe("RPT:HashMap PROBLEM ATTACHMENT RESULTS
            =" + paResults);
        }
    } catch (Exception e) {
        Z.log.writeToLog(Log.ERROR, "Error in method getSelect:" + e);
//ErrorWriter.write(e, ErrorWriter.LOG);
        throw e;
    } finally {
        if (paStmt != null) paStmt.close();
    }
}

/*****
 * Execute the query -- RELATIONSHIP_GROUP_PROBLEM = Driver Table
 *****/

if (doRGQuery) {
    try {
        if (rgKeys == null) rgKeys = new ArrayList();
        if (rgKeys.size() == 0) rgResults = new HashMap();
        else {
            rgStmt = conn.prepareStatement(rgQuery);

```

Report.java

```

/* Bind the block of problem IDs
-----*/
        for (int k = 0; k < rgKeys.size(); k++)
            rgStmt.setString(k + 1, (String) rgKeys.get(k));

/* Execute the query and process the result set
-----*/
        rgRs = rgStmt.executeQuery();
        rgResults = processResults(rgRs,
            rgDataAlias,
            ddeDisplayedDates,
            ddeTextFields,
            ddeDisplayURLs,
            dictionaryFields,
            session);

//Z.probe("RPT:HashMap RELATIONSHIP_GROUP_PROBLEM RESULTS =" + rgResults);
    }
    } catch (Exception e) {
        Z.log.writeToLog(Log.ERROR, "Error in method getSelect:" + e);
//ErrorWriter.write(e, ErrorWriter.LOG);
        throw e;
    } finally {
        if (rgStmt != null) rgStmt.close();
    }
}

/*****
 * Execute the query -- TEXT AREA, LOG AREA UDFs
*****/
String multiUdfSeparator =
Z.appDefaults.getAttribute("MULTIPLE_FIELD_SEPARATOR");

if (doUDFTextQuery || (doReleaseUdfTextQuery && doPRQuery)) {
    try {
//Z.probe("RPT:In Report.GetSelect - if doUDFTextQuery = true");

        boolean gotHash = false;
        boolean getNewList = false;

        int loopCnt = 0;
        Object tempObj = null;

        pKey = "";
        String ptKey = "";
        String pKeyPrev = "";
        String ptKeyPrev = "";
        String udfTextSqlString = "";
        String tempResult = "";

        StringBuffer udfText = new StringBuffer();
        String udfTimestamp = "";
        String udfUser = "";
        String udfName = "";
        String udfTextPrev = "";
        String udfTimestampPrev = "";
        String udfUserPrev = "";
        String udfNamePrev = "";
        String inUdfCriteria = "";

```

Report.java

```

Iterator udfRowIt = null;

HashMap tempUdfHash = null;
ArrayList tempTextList = new ArrayList();
ArrayList tempTimestampList = new ArrayList();
ArrayList tempUserList = new ArrayList();
ArrayList distinctUdfNameList = new ArrayList();
HashMap distinctUdfNames = new HashMap();

HashMap udfItem = new HashMap();

boolean multiUdfParents[] =
{doUDFTextQuery,doReleaseUdfTextQuery};

Z.probe("RPT:In Report.GetSelect - doUDFMultiQuery");

// Process UDFS for each of the different Parent Tables (currently Items &
// Releases)
//-----
for (int x = 0; x < multiUdfParents.length; x++){
    if (!multiUdfParents[x]) continue;
    if (x == 0){
        doUDFMultiQuery = true;
        doReleaseUdfTextQuery = false;
    }
    if (x == 1){
        doUDFMultiQuery = false;
        doReleaseUdfTextQuery = true;
    }
}

// Loop through each of the UDF's individually. Append each one to
// pRows in pResults given a problem id
//-----

if (doReleaseUdfTextQuery){
    doUDFTextQuery = false;

    if (!doPRQuery && !doReleaseUdfTextQuery) prResults =
new HashMap();

    inUdfCriteria = rInCriteria.toString();

    udfTextsSqlString = " select item_group.item2_id " +
        ", item_text.item_id PROBLEM_RELEASE_ID " +
        ", udf.name UDF_NAME " +
        ", item_text.text TEXT " +
        ", item_text.text_seq TEXT_SEQ " +
        ", item_text.id_seq " + problemUdfTextKey +
        ", item_text.timestamp TIMESTAMP " +
        ", item_text.user_id USER_ID " +
        ", security_user.first_name FIRST_NAME " +
        ", security_user.last_name LAST_NAME " +
        ", security_user.email EMAIL " +
        " from security_user, udf, item_text, item_group

" +
inUdfCriteria.toString() +
        " where item_text.item_id in " +
        " and udf.name = ? " +

```

```

Report.java
" and item_group.item1_id = item_text.item_id "
+
" and item_text.user_id =
security_user.security_user_id " +
" and udf.udf_id = item_text.udf_id " +
item_text.item_id, item_text.id_seq asc, item_text.text_seq asc ";
" order by item_group.item2_id,
}
else if (doUDFTextQuery){
doReleaseUdfTextQuery = false;
if (!doPQuery && !doUDFTextQuery) pResults = new
HashMap();
inUdfCriteria = inCriteria.toString();
udfTextSqlString = " select item_text.item_id " +
PROBLEM_KEY+
", udf.name UDF_NAME " +
", item_text.text TEXT " +
", item_text.text_seq TEXT_SEQ " +
", item_text.id_seq " + problemUdfTextKey +
", item_text.timestamp TIMESTAMP " +
", item_text.user_id USER_ID " +
", security_user.first_name FIRST_NAME " +
", security_user.last_name LAST_NAME " +
", security_user.email EMAIL " +
" from security_user, udf, item_text " +
" where item_text.item_id in " +
inUdfCriteria.toString() +
" and udf.name = ? " +
" and item_text.user_id =
security_user.security_user_id " +
" and udf.udf_id = item_text.udf_id " +
asc, item_text.text_seq asc ";
" order by item_text.item_id, item_text.id_seq
}
Z.probe(" udfTextSqlString " + udfTextSqlString);
Z.probe(" inUdfCriteria.toString() " +
inUdfCriteria.toString());
tempUdfHash = new HashMap();
tempTextList = new ArrayList();
tempTimestampList = new ArrayList();
tempUserList = new ArrayList();
// PROCESS ALL THE MULTI_VALUE UDFS
//-----
int udfSize = 0;
if (doUDFTextQuery) udfSize = udfTextFields.size();
else udfSize = releaseUdfTextFields.size();
for (int i = 0; i < udfSize; i++) {
tempUdfHash = new HashMap();
pKey = "";
pKeyPrev = "";
ptKeyPrev = "";
udfText.setLength(0);
udfTimestamp = "";
udfUser = "";
udfName = "";

```


Report.java

```

/* Completed processing a problem_id - each problem (pKey) has lists of comment
fields
-----*/
---*/

        if (/*!pKey.equals(pKeyPrev) && */!gotHash) {

/* Add to the current HashMap if its there
-----*/
        if (!pKeyPrev.equalsIgnoreCase("") && ((tempObj
= udfItem.get(pKeyPrev)) != null)) {
            tempUdfHash = (HashMap) tempObj;
//Z.probe("RPT:++FOUND HASHMAP FOR probelm "+ pKeyPrev + " : " + tempUdfHash);
        } else {
            tempUdfHash = new HashMap();
//Z.probe("RPT:++NOT FOUND HASHMAP FOR probelm "+ pKeyPrev + " : " + tempUdfHash);
        }
        gotHash = true;
        getNewList = true;
    }

/*    else if ( ((tempObj = udfItem.get(pKeyPrev)) != null) && !gotHash ){
tempUdfHash = (HashMap) tempObj;
//Z.probe("RPT:FOUND HASHMAP FOR probelm "+ pKeyPrev + " : " + tempUdfHash);
gotHash = true;
getNewList = true;
}
else{
//Z.probe("RPT:ALREADY HAVE HASHMAP FOR probelm "+ pKeyPrev + " : " + tempUdfHash);
}
*/

        if (getNewList) {

/* Use the current list if you've already processed that UDF
-----*/
        if ((tempObj = tempUdfHash.get(udfNamePrev +
_TEXT)) != null)
            tempTextList = (ArrayList) tempObj;
        else tempTextList = new ArrayList();

        if ((tempObj = tempUdfHash.get(udfNamePrev +
_TIMESTAMP)) != null)
            tempTimestampList = (ArrayList) tempObj;
        else tempTimestampList = new ArrayList();

        if ((tempObj = tempUdfHash.get(udfNamePrev +
_USER)) != null)
            tempUserList = (ArrayList) tempObj;
        else tempUserList = new ArrayList();
        getNewList = false;
    }

/* Merge all 4k blocks - each comment has its own ptKey
-----*/
        tempResult = udfRs.getString("TEXT");
        if ( tempResult != null &&
            Integer.parseInt(udfRs.getString("TEXT_SEQ")) >
1)
            tempResult =
tempResult.substring(Z.TEXT_OVERLAP);

```

```

Report.java
if ( ptKey.equals(ptKeyPrev) &&
(pKey.equals(pKeyPrev)) ){
    if (tempResult == null)
        udfText.append(emptyStringSpacer);
    else if (isTextOutputType())
        udfText.append(this.isTextOutputType()
                        ? tempResult
                        :
TextManager.convertUrl(getPrintableResult(tempResult)));
    else{
        if
("PRINTTEXT".equalsIgnoreCase(ddeDisplayType)){
            udfText.append(this.isTextOutputType()
                            ? tempResult
                            :
getPrintableResult(tempResult));
        }
        else{
            udfText.append(this.isTextOutputType()
                            ? tempResult
                            :
TextManager.convertUrl(getPrintableResult(tempResult)));
        }
    }
}
else{
    if
("PRINTTEXT".equalsIgnoreCase(ddeDisplayType))
        udfText = new
StringBuffer(this.isTextOutputType()
                ? tempResult
                :
getPrintableResult(tempResult));
    else
        udfText = new
StringBuffer(this.isTextOutputType()
                ? tempResult
                :
TextManager.convertUrl(getPrintableResult(tempResult)));
    String tempDate = formatDateString( dbt,
                                         su,
Convert.toCalendar(udfRs.getTimestamp("TIMESTAMP")),
                    emptyStringSpacer);
    udfTimestamp = getPrintableResult(tempDate);
    udfUser =
this.getEmailLink(udfRs.getString("USER_ID"),
                    udfRs.getString("FIRST_NAME"),
                    udfRs.getString("LAST_NAME"),
                    udfRs.getString("EMAIL"));
}

// a new item_text_id means a new comment so add the last comment info
// -----

if (!ptKeyPrev.equals(""))
    && (!ptKey.equals(ptKeyPrev) ||
!pKey.equals(pKeyPrev))

```

```

Report.java
&& gotHash) {

tempTextList.add(udfTextPrev);
tempTimestampList.add(udfTimestampPrev);
tempUserList.add(udfUserPrev);

//      if (!pKey.equals(pKeyPrev)){
tempUdfHash.put(udfNamePrev + _TEXT,
tempTextList);
tempUdfHash.put(udfNamePrev + _TIMESTAMP,
tempTimestampList);
tempUdfHash.put(udfNamePrev + _USER,
tempUserList);

udfItem.put(pKeyPrev, tempUdfHash);
//Z.probe("RPT:new ptkey = "+ ptKey +" >> pKey = "+pKey+" processed " + udfName + "
udfItem.put(pKeyPrev = " +pKeyPrev + ", > " + tempUdfHash);

gotHash = false;

// }
    }
} // end while rs

// track the distinct UDF Names
if (distinctUdfNames.get(udfName) == null) {
    distinctUdfNames.put(udfName, "");
    distinctUdfNameList.add(udfName + _TEXT);
    distinctUdfNameList.add(udfName + _TIMESTAMP);
    distinctUdfNameList.add(udfName + _USER);
}

/* Deal with the last record
-----*/
if (pKey.equals(pKeyPrev) && gotHash) {

//Z.probe("RPT:DOING LAST RECORD !!!!>>> tempUdfHash: " +tempUdfHash);
/* Use the current list if you've already processed that UDF
-----*/
if ((tempObj = tempUdfHash.get(udfName + _TEXT)) !=
null)

    tempTextList = (ArrayList) tempObj;
else tempTextList = new ArrayList();

if ((tempObj = tempUdfHash.get(udfName +
_TIMESTAMP)) != null)

    tempTimestampList = (ArrayList) tempObj;
else tempTimestampList = new ArrayList();

if ((tempObj = tempUdfHash.get(udfName + _USER)) !=
null)

    tempUserList = (ArrayList) tempObj;
else tempUserList = new ArrayList();
getNewList = false;

tempTextList.add(udfText.toString());
tempTimestampList.add(udfTimestamp);
tempUserList.add(udfUser);

tempUdfHash.put(doReleaseUdfTextQuery ?
"PROBLEM_RELEASE_ID" : PROBLEM_KEY_ALIAS, pKey);
tempUdfHash.put(udfName + _TEXT, tempTextList);
tempUdfHash.put(udfName + _TIMESTAMP,
tempTimestampList);

```

```

Report.java
tempUdfHash.put(udfName + _USER, tempUserList);
udfItem.put(pKey, tempUdfHash);
gotHash = false;
    }
    else {
// Add to the current HashMap if its there
//-----
        if ((tempObj = udfItem.get(pKey)) != null)
tempUdfHash = (HashMap) tempObj;
        else tempUdfHash = new HashMap();

/* Use the current list if you've already processed that UDF
-----*/
        if ((tempObj = tempUdfHash.get(udfName + _TEXT)) !=
null)
            tempTextList = (ArrayList) tempObj;
        else tempTextList = new ArrayList();

        if ((tempObj = tempUdfHash.get(udfName +
_TIMESTAMP)) != null)
            tempTimestampList = (ArrayList) tempObj;
        else tempTimestampList = new ArrayList();

        if ((tempObj = tempUdfHash.get(udfName + _USER)) !=
null)
            tempUserList = (ArrayList) tempObj;
        else tempUserList = new ArrayList();
        getNewList = false;

        tempTextList.add(udfText.toString());
        tempTimestampList.add(udfTimestamp);
        tempUserList.add(udfUser);

        tempUdfHash.put(doReleaseUdfTextQuery ?
"PROBLEM_RELEASE_ID" : PROBLEM_KEY_ALIAS, pKey);
        tempUdfHash.put(udfName + _TEXT, tempTextList);
        tempUdfHash.put(udfName + _TIMESTAMP,
tempTimestampList);
        tempUdfHash.put(udfName + _USER, tempUserList);

        udfItem.put(pKey, tempUdfHash);
        gotHash = false;
    }
//Z.probe("RPT:FINAL RECORD new pkey>> processed " + udfName + "
udfItem.put(pKeyPrev = " + pKey + ", > " + tempUdfHash);
//Z.probe("RPT:>>>> udfItem is : " + udfItem);
    }// end while

// Loop through the UDF text results and append each one
// to the pRows name/value hashmap in pResults given a problem id
// -----

        HashMap udfVals = new HashMap();
        String udfLayoutName = "";
        Iterator tagIt = null;

        ArrayList emptyList = new ArrayList();
        emptyList.add(this.emptyStringSpacer);

// pQuery already run

```

```

// -----
if ((doUDFTextQuery && doPQuery) || (doReleaseUdfTextQuery
&& doPRQuery)) {
    Z.probe("<< Adding to existing results >> ");
    Z.probe("<< Existing results - prResults : " +
prResults);
    Z.probe("<< Existing udfItem - udfItem : " + udfItem);

    // all ids in problem
    Iterator it = null;

    if (doUDFMultiQuery) it = pResults.keySet().iterator();
    else it = prResults.keySet().iterator();

    ArrayList multiReleases = null;
    ArrayList multiList = null;
    ArrayList tmpLst = null;
    StringBuffer multiSb = new StringBuffer();

    String prKey = "";

    while (it.hasNext()) {
        pRows = new HashMap();
        pKey = (String) it.next();

        if (doUDFTextQuery) pRows = (HashMap)
pResults.get(pKey);
        else{
            pRows = (HashMap) prResults.get(pKey);
            multiList = new ArrayList();
        }

        //
        //
        //
        //
        Z.probe(" The results pKey is : " + pKey);
        Z.probe(" This pKey has the following hash : " +
        Z.probe(" The udfItem hashmap is udfItem : " +
        udfItem);

        // need to loop through each of the releases for any given problem key pKey
        //-----
        if (doReleaseUdfTextQuery){
            multiReleases = (ArrayList)
pRows.get("PROBLEM_RELEASE_ID");

            for (int i = 0; i < multiReleases.size(); i++){
                prKey = (String) multiReleases.get(i);

                if ((udfVals = (HashMap) udfItem.get(prKey))
== null) {

                    for (int j = 0; j <
distinctUdfNameList.size(); j++)
                        pRows.put((String)
distinctUdfNameList.get(j), emptyList);

                    // pRows.put(this.problemKeyAlias, prKey);

                }
            }
        }
    }
}
else {

```

Report.java

```

distinctUdfNameList.size(); j++) {
    distinctUdfNameList.get(j);
    null)
    emptyList);

    udfVals.get(udfLayoutName);

    tmpLst.size(); y++){
        multiSb.append(multiUdfSeparator);
        multiSb.append((String)tmpLst.get(y));

        multiSb.toString());
        multiList);

        // pRows.put(problemKeyAlias, prKey);
    }

    }

    prResults.put(pKey, pRows);

    }
    else{
        if ((udfVals = (HashMap) udfItem.get(pKey)) ==
        null) {

            distinctUdfNameList.size(); i++)
            distinctUdfNameList.get(i), emptyList);

            pRows.put(PROBLEM_KEY_ALIAS, pKey);

        }
        else {
            for (int i = 0; i <
                udfLayoutName = (String)
                if (udfVals.get(udfLayoutName) == null)
                    pRows.put(udfLayoutName, emptyList);
                else
                    pRows.put(udfLayoutName, (ArrayList)
                    udfVals.get(udfLayoutName));
            }
        }
    }
}

for (int j = 0; j <
    udfLayoutName = (String)
    if (udfVals.get(udfLayoutName) ==
        pRows.put(udfLayoutName,
    else{
        tmpLst = (ArrayList)
        //tmpD
        multiSb.setLength(0);
        for (int y = 0; y <
            if (y >0)

        }
        multiList.add(
        pRows.put(udfLayoutName,
    }
} // end for
}

// pRows.put(problemKeyAlias, prKey);
}

}

prResults.put(pKey, pRows);

}
else{
    if ((udfVals = (HashMap) udfItem.get(pKey)) ==
    null) {

        distinctUdfNameList.size(); i++)
        distinctUdfNameList.get(i), emptyList);

        pRows.put(PROBLEM_KEY_ALIAS, pKey);

    }
    else {
        for (int i = 0; i <
            udfLayoutName = (String)
            if (udfVals.get(udfLayoutName) == null)
                pRows.put(udfLayoutName, emptyList);
            else
                pRows.put(udfLayoutName, (ArrayList)
                udfVals.get(udfLayoutName));
        }
    }
}

udfVals.get(udfLayoutName));

```



```

Report.java
    }
    pRows.put(PROBLEM_KEY_ALIAS, pKey);
}

pResults.put(pKey, pRows);
}
}
}

// Never ran p_query use sorted keys
// -----
else {
    Z.probe("<< Adding to new results >> ");
    for (int k = 0; k < sortedKeys.size(); k++) {

        pRows = new HashMap();
        pKey = (String) sortedKeys.get(k);
        if ((udfVals = (HashMap) udfItem.get(pKey)) == null)

        {

            for (int i = 0; i < distinctUdfNameList.size();
i++)
                pRows.put((String)
distinctUdfNameList.get(i), emptyList);

            pRows.put(PROBLEM_KEY_ALIAS, pKey);
        } else {
            for (int i = 0; i < distinctUdfNameList.size();
i++) {

                udfLayoutName = (String)
distinctUdfNameList.get(i);

                if (udfVals.get(udfLayoutName) == null)
                    pRows.put(udfLayoutName, emptyList);
                else
                    pRows.put(udfLayoutName, (ArrayList)
udfVals.get(udfLayoutName));

            }
            pRows.put(PROBLEM_KEY_ALIAS, pKey);
        }
        if (doUDFMultiQuery) pResults.put(pKey, pRows);
        else prResults.put(pKey, pRows);
    }
}
}

}

//
//
/*
while (it.hasNext()) {
pRows = new HashMap();
pKey = (String) it.next();
pRows = (HashMap) pResults.get(pKey);

if ((udfVals = (HashMap) udfItem.get(pKey)) == null) {

for (int i = 0; i < distinctUdfNameList.size(); i++)
pRows.put((String) distinctUdfNameList.get(i), emptyList);

pRows.put(this.problemKeyAlias, pKey);
}
}

```

```

else {
    for (int i = 0; i < distinctUdfNameList.size(); i++) {
        udfLayoutName = (String) distinctUdfNameList.get(i);
        if (udfVals.get(udfLayoutName) == null)
            pRows.put(udfLayoutName, emptyList);
        else
            pRows.put(udfLayoutName, (ArrayList) udfVals.get(udfLayoutName));
    }
    pRows.put(problemKeyAlias, pKey);
    pResults.put(pKey, pRows);
}

/// Never ran p_query use sorted keys
///-----
else {
    for (int k = 0; k < sortedKeys.size(); k++) {

        pRows = new HashMap();
        pKey = (String) sortedKeys.get(k);

        if ((udfVals = (HashMap) udfItem.get(pKey)) == null) {
            for (int i = 0; i < distinctUdfNameList.size(); i++)
                pRows.put((String) distinctUdfNameList.get(i), emptyList);

            pRows.put(this.problemKeyAlias, pKey);
        }
        else {
            for (int i = 0; i < distinctUdfNameList.size(); i++) {

                udfLayoutName = (String) distinctUdfNameList.get(i);
                if (udfVals.get(udfLayoutName) == null)
                    pRows.put(udfLayoutName, emptyList);
                else
                    pRows.put(udfLayoutName, (ArrayList) udfVals.get(udfLayoutName));
            }
            pRows.put(problemKeyAlias, pKey);
            pResults.put(pKey, pRows);
        }
    }
}

*/
        catch (Exception e) {
            Z.log.writeToLog(Log.ERROR, "Error in method getSelect:" + e);
//ErrorWriter.write(e, ErrorWriter.LOG);
            throw e;
        } finally {
            if (udfStmt != null) udfStmt.close();
            Z.log.writeToLog(Log.DEBUG3, this,
"^^^^Report.getSelect.doUDFTextQuery = true udfTEXTStatement closed");
//Z.probe("RPT:In Report.GetSelect - if doUDFTextQuery = true --> pResults : "+
pResults);
        }
    }

}

//*****
//*   Execute the query  -- MULTI ROW UDFs  -- MULTI UDFs
//*****

    if (doUDFMultiQuery || (doReleaseUdfMultiQuery && doPRQuery)) {

```

```

Report.java
boolean gotHash = false;
boolean getNewList = false;
int loopCnt = 0;
Object tempObj = null;

pKey = "";
String udfMultisqlString = "";
String tempResult = "";
String udfName = "";
String udfListTitle = "";
String udfListId = "";
String pKeyPrev = "";
String udfListTitlePrev = "";
String udfNamePrev = "";

Iterator udfRowIt = null;

HashMap tempUdfHash = null;
ArrayList tempTitleList = new ArrayList();
ArrayList tempIdList = new ArrayList();
HashMap distinctUdfNames = new HashMap();
ArrayList distinctUdfNameList = new ArrayList();

HashMap udfItem = new HashMap();
String inUdfCriteria = "";
String stdUdfMultisql = "";

boolean multiUdfParents[] =
{doUDFMultiQuery,doReleaseUdfMultiQuery};

try {

    Z.probe("RPT:In Report.GetSelect - doUDFMultiQuery");

//    Process UDFS for each of the different Parent Tables (currently Items &
//    Releases)
//-----
    for (int x = 0; x < multiUdfParents.length; x++){
        if (!multiUdfParents[x]) continue;
        if (x == 0){
            doUDFMultiQuery = true;
            doReleaseUdfMultiQuery = false;
        }
        if (x == 1){
            doUDFMultiQuery = false;
            doReleaseUdfMultiQuery = true;
        }
    }

// Loop through each of the UDF's individually. Append each one to
// pRows in pResults given a problem id
//-----

        if (doReleaseUdfMultiQuery){
            if (!doPRQuery && !doReleaseUdfMultiQuery) prResults =
new HashMap();
            inUdfCriteria = rInCriteria.toString();
            stdUdfMultisql =
            " select ig.item2_id " + PROBLEM_KEY + ",
iu.item_id PROBLEM_RELEASE_ID , " +

```



```

                                Report.java
                                udfStmt = conn.prepareStatement(udfMultiSqlString);

/* Bind the block of problem IDs and then bind UDF Name & Execute the query
-----*/
                                if (doUDFMultiQuery){
                                    for (int k = 0; k < sortedKeys.size(); k++)
                                        udfStmt.setString(k + 1, (String)
sortedKeys.get(k));

                                    udfStmt.setString(sortedKeys.size() + 1, udfName);
                                }
                                else{
                                    for (int k = 0; k < releaseKeys.size(); k++)
                                        udfStmt.setString(k + 1, (String)
releaseKeys.get(k));

                                    udfStmt.setString(releaseKeys.size() + 1, udfName);
                                }

                                Z.probe("DOING MULTI UDF _ About to execute
udfMultiSqlString = " + udfMultiSqlString);

                                udfRs = udfStmt.executeQuery();

// Process the result rest
// -----
                                while (udfRs.next()) {

// To compare to the last processed_value
// -----
                                    udfListTitlePrev = udfListTitle;
                                    udfNamePrev = udfName;
                                    pKeyPrev = pKey;

// Get the keys
// -----
                                udfRs.getString(PROBLEM_KEY); // the problem id
                                // the release id
                                    else pKey = udfRs.getString("PROBLEM_RELEASE_ID");

                                    udfName = udfRs.getString("UDF_NAME");

//Z.probe("RPT:=====
=====");
//Z.probe("RPT:>> processing UdfName " + udfName + " (pKey = " + pKey + "
(pkeyPrev = "+ pKeyPrev +") ");

// Completed processing a problem_id - each problem (pKey) has lists of comment
fields
// -----
                                udfListTitle =
(getPrintableResult(udfRs.getString("Title")));

                                    if (!pKey.equals(pKeyPrev) && !gotHash) {

// Add to the current HashMap if its there
// -----
                                    if (!pKeyPrev.equalsIgnoreCase("")) && ((tempObj

```

```

Report.java
= udfItem.get(pKeyPrev)) != null)) {
    tempUdfHash = (HashMap) tempObj;
    ////Z.probe("RPT:GOT HASH :"+ gotHash+" FOUND HASHMAP FOR probleM "+ pKeyPrev + " :
    " + tempUdfHash);
    }
    else {
        tempUdfHash = new HashMap();
        ////Z.probe("RPT:GOT HASH :"+ gotHash+" NOT FOUND HASHMAP FOR probleM "+ pKeyPrev +
        " : " + tempUdfHash);
    }
    gotHash = true;
    getNewList = true;
}
else if (((tempObj = udfItem.get(pKeyPrev)) != null)
&& !gotHash) {
    tempUdfHash = (HashMap) tempObj;
    ////Z.probe("RPT:GOT HASH :"+ gotHash+" FOUND HASHMAP FOR probleM "+ pKeyPrev + "
    : " + tempUdfHash);
    gotHash = true;
    getNewList = true;
}
// else if (gotHash){
// //Z.probe("RPT:GOT HASH :"+ gotHash+" ALREADY GOT HASHMAP FOR probleM "+
// pKeyPrev + " : " + tempUdfHash);
// }

    if (!pKeyPrev.equals("")) {

//Z.probe("RPT:ADDING udfList TitlePrev for pkeyPrev: "+ pKeyPrev + " - " +
udfListTitlePrev );
        tempTitleList.add(udfListTitlePrev);

/* a new problem_id means a new arraylist
-----*/
        if (!pKey.equals(pKeyPrev) && gotHash) {
            tempUdfHash.put(udfNamePrev, tempTitleList);
            udfItem.put(pKeyPrev, tempUdfHash);
            gotHash = false;
            tempTitleList = new ArrayList();
            ////Z.probe("RPT:NOW SET GOT HASH TO " + gotHash + " new pKey = "+pKey+" processed "
            + udfName + " udfItem.put(pKeyPrev = " +pKeyPrev + ", > " + tempUdfHash);
        }
    } // end while rs

// track the distinct UDF Names
    if (distinctUdfNames.get(udfName) == null)
distinctUdfNameList.add(udfName);

//Z.probe("RPT:FINAL RESULTS pKeyPrev -"+ pKeyPrev + " pKey-" + pKey + "
gotHash-"+gotHash);

// Deal with the last record
//-----
    if (pKey.equals(pKeyPrev)) {
        if (!gotHash) tempUdfHash = new HashMap();
        tempTitleList.add(udfListTitle);
        tempUdfHash.put( doReleaseUdfMultiQuery ?
"PROBLEM_RELEASE_ID" : PROBLEM_KEY_ALIAS, pKey);
        tempUdfHash.put(udfName, tempTitleList);
    }

```

```

                                Report.java
                                udfItem.put(pKey, tempUdfHash);
                                }
                                else {

// Add to the current HashMap if its there
//-----
                                if ((tempObj = udfItem.get(pKey)) != null)
tempUdfHash = (HashMap) tempObj;
                                else tempUdfHash = new HashMap();

                                tempTitleList = new ArrayList();
                                tempTitleList.add(udfListTitle);

                                tempUdfHash.put( doReleaseUdfMultiQuery ?
"PROBLEM_RELEASE_ID" : PROBLEM_KEY_ALIAS, pKey);
                                tempUdfHash.put(udfName, tempTitleList);

                                udfItem.put(pKey, tempUdfHash);
                                }
                                gotHash = false;
                                tempTitleList = new ArrayList();

//Z.probe("RPT:FINAL RECORD new pkey>> processed " + udfName + "
udfItem.put(pKeyPrev = " + pKey + ", > " + tempUdfHash);
//Z.probe("RPT:>>>> udfItem is : " + udfItem);
                                }// end for

/* Loop through the UDF text results and append each one
to the pRows name/value hashmap in pResults given a problem id
-----*/

                                HashMap udfVals = new HashMap();
                                String udfLayoutName = "";
                                Iterator tagIt = null;

                                ArrayList emptyList = new ArrayList();
                                emptyList.add(this.emptyStringSpacer);

// pQuery already run
// -----
                                Z.probe(" About to add multi-udf to results " );

                                if ((doUDFMultiQuery && doPQuery) || (doReleaseUdfMultiQuery
&& doPRQuery)) {
                                Z.probe("<< Adding to existing results >> ");
                                Z.probe("<< Existing results - prResults : " +
prResults);
                                Z.probe("<< Existing udfItem - udfItem : " + udfItem);
                                // all ids in problem
                                Iterator it = null;

                                if (doUDFMultiQuery) it = pResults.keySet().iterator();
                                else it = prResults.keySet().iterator();

                                ArrayList multiReleases = null;
                                ArrayList multiList = null;
                                ArrayList tmpLst = null;
                                StringBuffer multiSb = new StringBuffer();

                                String prKey = "";

```



```

Report.java

while (it.hasNext()) {
    pRows = new HashMap();
    pKey = (String) it.next();

    if (doUDFMultiQuery) pRows = (HashMap)
pResults.get(pKey);
    else{
        pRows = (HashMap) prResults.get(pKey);
        multiList = new ArrayList();
    }

    //          Z.probe(" The results pKey is : " + pKey);
    //          Z.probe(" This pKey has the following hash : " +
pRows);
    //          Z.probe(" The udfItem hashmap is udfItem : " +
udfItem);

    // need to loop through each of the releases for any given problem key pKey
    //-----
    if (doReleaseUdfMultiQuery){
        multiReleases = (ArrayList)
pRows.get("PROBLEM_RELEASE_ID");

        for (int i = 0; i < multiReleases.size(); i++){
            prKey = (String) multiReleases.get(i);

            if ((udfVals = (HashMap) udfItem.get(prKey))
== null) {

                for (int j = 0; j <
distinctUdfNameList.size(); j++)
                    pRows.put((String)
distinctUdfNameList.get(j), emptyList);
                //    pRows.put(this.problemKeyAlias, prKey);
            }
            else {
                for (int j = 0; j <
distinctUdfNameList.size(); j++) {
                    udfLayoutName = (String)
distinctUdfNameList.get(j);
                    if (udfVals.get(udfLayoutName) ==
null)
                        pRows.put(udfLayoutName,
emptyList);
                    else{
                        tmpLst = (ArrayList)
udfVals.get(udfLayoutName);
                        multiSb.setLength(0);
                        for (int y = 0; y <
tmpLst.size(); y++){
                            if (y >0)
                                multiSb.append(multiUdfSeparator);
                                multiSb.append((String)tmpLst.get(y));
                        }
                    }
                }
            }

            Z.log.writeToLog(Z.log.DEBUG5,"ABOut to append multiSb to the release multiUDF " +
multiSb.toString());
        }
    }
}

```

```

        multiList.add(
        multiList.toString());
        multiList);

        }
    } // end for

    // pRows.put(problemKeyAlias, prKey);
    }

    }
    prResults.put(pKey, pRows);

    }
    else{
        if ((udfVals = (HashMap) udfItem.get(pKey)) ==
        null) {
            for (int i = 0; i <
            distinctUdfNameList.size(); i++)
                pRows.put((String)
                distinctUdfNameList.get(i), emptyList);

                pRows.put(PROBLEM_KEY_ALIAS, pKey);
            }
            else {
                for (int i = 0; i <
                distinctUdfNameList.size(); i++) {
                    distinctUdfNameList.get(i);
                    if (udfVals.get(udfLayoutName) == null)
                        pRows.put(udfLayoutName, emptyList);
                    else
                        pRows.put(udfLayoutName, (ArrayList)
                        udfVals.get(udfLayoutName));
                }
                pRows.put(PROBLEM_KEY_ALIAS, pKey);
            }
            prResults.put(pKey, pRows);
        }
    }
}

// Never ran p_query use sorted keys
// -----
    else {
        Z.probe("<< Adding to new results >> ");
        for (int k = 0; k < sortedKeys.size(); k++) {
            pRows = new HashMap();
            pKey = (String) sortedKeys.get(k);
            if ((udfVals = (HashMap) udfItem.get(pKey)) == null)
            {
                for (int i = 0; i < distinctUdfNameList.size();

```

```

Report.java

i++)
distinctUdfNameList.get(i), emptyList);

pRows.put((String)
pRows.put(PROBLEM_KEY_ALIAS, pKey);
} else {
for (int i = 0; i < distinctUdfNameList.size();
i++) {

udfLayoutName = (String)
distinctUdfNameList.get(i);
if (udfVals.get(udfLayoutName) == null)
pRows.put(udfLayoutName, emptyList);
else
pRows.put(udfLayoutName, (ArrayList)
udfVals.get(udfLayoutName));
}
pRows.put(PROBLEM_KEY_ALIAS, pKey);
}
if (doUDFMultiQuery) pResults.put(pKey, pRows);
else prResults.put(pKey, pRows);
}
}
}
}
catch (Exception e) {
Z.log.writeToLog(Log.ERROR, "Error in method getSelect:" + e);
//ErrorWriter.write(e, ErrorWriter.LOG);
throw e;
} finally {
if (udfstmt != null) udfstmt.close();
Z.log.writeToLog(Log.DEBUG, this,
"^^^^Report.getSelect.doUDFMultiQuery = true udfMULTIStatement closed");
//Z.probe("RPT:In Report.GetSelect - if doUDFMultiQuery = true --> pResults : "+
pResults);
}

}

/*****
* SORT THE RESULTS BASED ON THE ORDER OF THE MASTER LIST
*
* -- create lists for the multi valued results
*
*
* *****/

HashMap nameValues = null; // the final hashmap for this problem id --
gets pmNameValues & prNameValues appended to it
HashMap pmNameValues = null;
HashMap prNameValues = null;
HashMap paNameValues = null;
HashMap rgNameValues = null;

HashMap prMultiLookups = null;
HashMap pmMultiLookups = null;
HashMap paMultiLookups = null;
HashMap rgMultiLookups = null;

ArrayList latestMultiList = null;
ArrayList tempList = null;
Object tmpObj = null;

```

Report.java

```

Iterator pi = null;
a record      Iterator pmi = null; // problem module iterator for name/value pairs in
a record      Iterator pri = null; // problem release iterator for name/value pairs in
record        Iterator pai = null; // attachment iterator for name/value pairs in a
in a record   Iterator rgi = null; // relationship group iterator for name/value pairs
String sortedId = "";

tempAlias = "";
boolean isList = false;

/* Combine the results given the SORTED ID -- combine the name/value hashMap
-----*/
for (int k = 0; k < sortedKeys.size(); k++) {

    nameValues = new HashMap();
    pmNameValues = new HashMap();
    prNameValues = new HashMap();
    paNameValues = new HashMap();
    rgNameValues = new HashMap();
    prMultiLookups = new HashMap();
    pmMultiLookups = new HashMap();
    paMultiLookups = new HashMap();
    paMultiLookups = new HashMap();

    sortedId = (String) sortedKeys.get(k);

/* Get the problem results back in the right order
-----*/
    tmpObj = null;
    if ((pResults != null)) tmpObj = pResults.get(sortedId);
    if (tmpObj != null)
        nameValues = (HashMap) tmpObj;
    else {

        for (int i = 0; i < problemFieldList.size(); i++) {
            key = (String) problemFieldList.get(i);
            thisDDE = (DDEntry) (dictionaryFields.get(key));

            ddeType = thisDDE.getType();
            ddeName = thisDDE.getName();

            if (nameValues.get(ddeName) == null) {
                if (thisDDE.isMultipleValue() &&
!ddeName.equalsIgnoreCase(PROBLEM_KEY_ALIAS))
                    nameValues.put(ddeName, new ArrayList());
                else if (ddeName.equalsIgnoreCase(PROBLEM_KEY_ALIAS))
                    nameValues.put(ddeName, sortedId);
                else
                    nameValues.put(ddeName, emptyStringSpacer);
            }
        }
    }

}

/* Process the Problem Module Results
-----*/
if (doPMQuery) {

```

```

Report.java
Z.probe( "ABOUT TO APPEND MODULE RESULTS TO FINAL RESULTS!!
pmResults : " + pmResults);
Z.probe( "pmResults.get("+sortedId+") = " +
pmResults.get(sortedId));
if ((tmpObj = pmResults.get(sortedId)) != null) { // if there
are results for this ID
    pmNameValues = (HashMap) tmpObj;
//Z.probe( "pmNameValues = " + pmNameValues);
/* Merge the results
-----*/
    pmi = pmNameValues.keySet().iterator();
    while (pmi.hasNext()) {

        isList = false;
        tempAlias = (String) pmi.next();

// check to see if its a list
        if ("Y".equalsIgnoreCase((String)
pmDataAlias.get(tempAlias))
            && !tempAlias.equalsIgnoreCase(PROBLEM_KEY_ALIAS))
            nameValues.put(tempAlias, (ArrayList)
pmNameValues.get(tempAlias));
        else
            nameValues.put(tempAlias, (String)
pmNameValues.get(tempAlias));
    }
} // end if there are results

/* If there are no results for this sorted ID, we still need to
send back an empty.
-----*/
else {
    for (int i = 0; i < moduleFieldList.size(); i++) {
        key = (String) moduleFieldList.get(i);
        thisDDE = (DDEntry) (dictionaryFields.get(key));

        ddeType = thisDDE.getType();
        ddeName = thisDDE.getName();

        if (nameValues.get(ddeName) == null) {
            if (thisDDE.isMultipleValue() &&
!ddeName.equalsIgnoreCase(PROBLEM_KEY_ALIAS))
                nameValues.put(ddeName, new ArrayList());
            else if
(ddeName.equalsIgnoreCase(PROBLEM_KEY_ALIAS))
                nameValues.put(ddeName, sortedId);
            else
                nameValues.put(ddeName, emptyStringSpacer);
        }
    }
}

} // end if doPMQuery

/* Process the Problem Release Results
-----*/

if (doPRQuery) {
    Z.probe( "ABOUT TO APPEND RELEASE RESULTS TO FINAL RESULTS!!");
    Z.probe( "prResults !! == " + prResults);
    Z.probe( "prResults.get("+sortedId+") = " +
prResults.get(sortedId));

```

Report.java

```

        if ((tmpObj = prResults.get(sortedId)) != null) { // if there
are results for this ID

                prNameValues = (HashMap) tmpObj;
//Z.probe( "prNameValues = " + prNameValues);

// add the release multi-udfs to the prDataAlias list
//-----
                for (int j = 0; j < releaseUdfMultiFields.size(); j++)
                        prDataAlias.put(releaseUdfMultiFields.get(j), "Y");

                for (int j = 0; j < releaseUdfTextFields.size(); j++){
                        prDataAlias.put(releaseUdfTextFields.get(j)+_TEXT, "Y");
                        prDataAlias.put(releaseUdfTextFields.get(j)+_USER, "Y");

prDataAlias.put(releaseUdfTextFields.get(j)+_TIMESTAMP, "Y");
                        prDataAlias.put(releaseUdfTextFields.get(j), "Y");
                }

                Z.log.writeToLog(Z.log.DEBUG5, " THE prDATAALIASES ARE :
" + prDataAlias);
// Merge the results
//-----

                pri = prNameValues.keySet().iterator();
                while (pri.hasNext()) {

                        isList = false;
                        tempAlias = (String) pri.next();

                        Z.log.writeToLog(Z.log.DEBUG5, " RELEASE RESULTS --
Processing tempAlias = " +tempAlias);

// check to see if its a list
                        if ("Y".equalsIgnoreCase((String)
prDataAlias.get(tempAlias))
                                && !tempAlias.equalsIgnoreCase(PROBLEM_KEY_ALIAS))
                                nameValues.put(tempAlias, (ArrayList)
prNameValues.get(tempAlias));
                        else
                                nameValues.put(tempAlias, (String)
prNameValues.get(tempAlias));

                } // end while pri hasNext() nameValues
        } // end if there are results for that ID

// If there are no release results for this sorted ID, we still need to
// send back an empty.
// -----
        else {
                Z.log.writeToLog(Z.log.DEBUG5, " RELEASE RESULTS -- no
release results for this sorted ID " +sortedId );
                Z.log.writeToLog(Z.log.DEBUG5, " RELEASE RESULTS -- no
release results releaseFieldList IS : "+ releaseFieldList);

                for (int i = 0; i < releaseFieldList.size(); i++) {
                        key = (String) (releaseFieldList.get(i));
                        thisDDE = (DDEntry) (dictionaryFields.get(key));
                        ddeType = thisDDE.getType();
                        ddeName = thisDDE.getName();

```

```

Report.java
    tempList = new ArrayList();
    tempList.add(emptyStringSpacer);
    if (nameValues.get(ddeName) == null) {
        if (thisDDE.isMultipleValue() &&
!ddeName.equalsIgnoreCase(PROBLEM_KEY_ALIAS))
            nameValues.put(ddeName, tempList);
        else if
(ddeName.equalsIgnoreCase(PROBLEM_KEY_ALIAS))
            nameValues.put(ddeName, sortedId);
        else
            nameValues.put(ddeName, emptyStringSpacer);
    }
}
//Z.probe("---NAME VALUES : " + nameValues);
}

/* Process the Attachment Results
-----*/

    if (doPAQuery) {
        if ((tmpObj = paResults.get(sortedId)) != null) { // if there
are results for this ID
            paNameValues = (HashMap) tmpObj;
            Z.log.writeToLog(Z.log.DEBUG5, " paNameValues = "
+paNameValues );
            // Merge the results
            // -----
            pai = paNameValues.keySet().iterator();
            while (pai.hasNext()) {

                isList = false;
                tempAlias = (String) pai.next();

                if ("Y".equalsIgnoreCase((String)
paDataAlias.get(tempAlias))
                    && !tempAlias.equalsIgnoreCase(PROBLEM_KEY_ALIAS)){
                    Z.log.writeToLog(Z.log.DEBUG5, " tempAlias = "
+tempAlias );
                    Z.log.writeToLog(Z.log.DEBUG5, "
/ (ArrayList) paNameValues.get(tempAlias) = " +(ArrayList)
paNameValues.get(tempAlias));
                    nameValues.put(tempAlias, (ArrayList)
paNameValues.get(tempAlias));
                }
                else
                    nameValues.put(tempAlias, (String)
paNameValues.get(tempAlias));
            }
        } // end if there are results

        /* If there are no results for this sorted ID, we still need to
send back an empty.
-----*/
        else {
            for (int i = 0; i < attachmentFieldList.size(); i++) {
                key = (String) attachmentFieldList.get(i);
                Z.probe("LOOKING AT ATTACHMENTS about to look for
ddentry with key = " + key);
                thisDDE = (DDEntry) (dictionaryFields.get(key));
                ddeType = thisDDE.getType();
                ddeName = thisDDE.getName();
            }
        }
    }
}

```


Report.java

```

        if (nameValues.get(ddeName) == null) {
            if (thisDDE.isMultipleValue() &&
!ddeName.equalsIgnoreCase(PROBLEM_KEY_ALIAS))
                nameValues.put(ddeName, new ArrayList());
            else if
(ddename.equalsIgnoreCase(PROBLEM_KEY_ALIAS))
                nameValues.put(ddeName, sortedId);
            else
                nameValues.put(ddeName, emptyStringSpacer);
        }
    }
} // end if doPAQuery

finalResults.put(sortedId, nameValues);
}
//Z.log.writeToLog(Z.log.DEBUG, "RPT:validation list FINAL RESULTS:" +
finalResults);
} catch (Exception e) {
    Z.log.writeToLog(Log.ERROR, this, "Error in method getSelect:" + e);
    ErrorWriter.write(e, ErrorWriter.LOG);
    throw e;
} finally {
    if (pstmt != null) pstmt.close();
    if (pmstmt != null) pmstmt.close();
    if (prstmt != null) prstmt.close();
    if (pastmt != null) pastmt.close();
    Z.log.writeToLog(Log.DEBUG3, this,
"^^^^Report.getSelect.doUDFMulti/TextQuery = true  udfMULTI/TEXTStatement closed");
    if (udfstmt != null) udfstmt.close();
    multiLookups.clear();

//
// PRINT OUT TO THE LOGS FOR TESTING PURPOSES
//-----
    Iterator itx = finalResults.keySet().iterator();
    Iterator itv = null;
    HashMap h = null;
    String id = "";
    String name = "";
    String value = "";
    Z.log.writeToLog(Log.DEBUG4, "THE FINAL RESULTS ARE: ");
    Z.log.writeToLog(Log.DEBUG4, "-----");
    while (itx.hasNext()){
        id = (String)itx.next();
        Z.log.writeToLog(Log.DEBUG4, "PROCESSING FOR ID: "+ id);
        h = (HashMap)finalResults.get(id);
        itv = h.keySet().iterator();
        while (itv.hasNext()){
            name = (String)itv.next();
            Z.log.writeToLog(Log.DEBUG4, "          " + name + "          /          "
+ h.get(name));
        }
    }
    return finalResults;

}

}

/**
 * getReportList()
 * @param dbconn

```

```

Report.java
* @return HashMap of ids as key and titles
*
* Gets the list of reports for that user
**/
public static HashMap getReportList(String userId, Connection dbConn) throws
Exception {
    Z.log.writeToLog(Log.DEBUG1, "Entering Report.getReportList Method");
    HashMap idRpts = new HashMap();
    String query = null;
    String id = null;
    String title = null;
    PreparedStatement statement = null;
    ResultSet rset = null;
    Report rpt = null;

    try {

        query = "SELECT report_id, title, report_type_name, description,
title_map_key " +
            "from REPORT where security_User_id = ? " +
            "and UPPER(report_type_name) not in ('QUICKLIST', 'DETAILED') " +
            "order by title ";
        statement = dbConn.prepareStatement(query);
        statement.setString(1, userId);
        String bv = PreparedStatementProxy.getBindValues(statement);
        Z.log.writeToLog(Log.DEBUG2, "Report.getReportList QUERY = " + query);
        Z.log.writeToLog(Log.DEBUG2, "Report.getReportList BindValues = " + bv);
        rset = statement.executeQuery();

        while (rset.next()) {
            rpt = new Report();
            rpt.setName(rset.getString("TITLE"));
            rpt.setReportType(rset.getString("REPORT_TYPE_NAME"));
            rpt.setTitle(rset.getString("DESCRIPTION"));
            rpt.titleMapKey = rset.getLong("TITLE_MAP_KEY");
            idRpts.put(rset.getString("REPORT_ID"), rpt);
        }
        return idRpts;
    } catch (Exception e) {
        Z.log.writeToLog(Log.ERROR, "Error in method getReportList:" + e);
        //ErrorWriter.write(e, ErrorWriter.LOG);
        throw e;
    } finally {
        if (statement != null) statement.close();
    }
}

/**
* buildQueryString()
* @param select String
* @param from String
* @param where String
* @param andProblem String
* @param andOther String
* @param andUDF String
* @param andKeyword String
* @return String the query string
*
* Gets a new report id from sequence
**/
private String buildQueryString(String select,

```

```

Report.java
String from,
String where,
String andPreCondition,
String andItemType,
String andProblem,
String andOther,
String andUDF,
String andKeyword,
String andSortOrder,
String orderBy,
String groupBy) throws Exception {

boolean whereExists = false;
boolean groupByExists = false;
StringBuffer query = new StringBuffer();
int defaultLength = 5; //" and ".length();

try {

    query.append(select + from + where);

    /**
     * ADJUST THE 1st " AND " in the where criteria
     * So if there is nothing in the where clause, the first statement in
the where clause
     * needs to have its first "and" string removed otherwise the sql
would read : "where and exists (select....)"
     */

    if (andPreCondition.length() > defaultLength) {
        whereExists = true;
        query.append("(" + andPreCondition + ")");
    }

    if (whereExists)
        query.append(andItemType);
    else if (andItemType.length() > defaultLength) {
        whereExists = true;
        query.append(andItemType.substring(4));
    }

    if (whereExists)
        query.append(andProblem);
    else if (andProblem.length() > defaultLength) {
        whereExists = true;
        query.append(andProblem.substring(4));
    }

    if (whereExists)
        query.append(andOther);
    else if (andOther.length() > defaultLength) {
        query.append(andOther.substring(4));
        whereExists = true;
    }

    if (whereExists)
        query.append(andUDF);
    else if (andUDF.length() > defaultLength) {
        query.append(andUDF.substring(4));
        whereExists = true;
    }
}

```

Report.java

```
        if (whereExists)
            query.append(andKeyword);
        else if (andKeyword.length() > defaultLength) {
            query.append(andKeyword.substring(4));
            whereExists = true;
        }

        if (whereExists)
            query.append(andSortOrder);
        else if (andSortOrder.length() > defaultLength) {
            query.append(andSortOrder.substring(4));
            whereExists = true;
        }

        if (groupBy.length() > " group by ".length()) {
            query.append(groupBy);
            groupByExists = true;
        }
    }
    catch (Exception e) {
        Z.log.writeToLog(Log.ERROR, "Error in method buildQueryString:" + e);
        throw e;
    }
    if (whereExists)
        return query.toString() + orderBy;
    else {
        if (groupByExists) return select + from + groupBy + orderBy;
        else return select + from + orderBy;
    }
}

/**
 * getNewReportId()
 * @param dbconn
 * @return new reportid
 *
 * Gets a new report id from sequence
 */
public static String getNewReportId() throws Exception {
    try {
        return Sequence.getNewSequenceId("REPORT_SEQ");
    } catch (Exception e) {
        throw e;
    }
}

/**
 * setConnection
 * @param dbconn
 *
 */
public void setConnection(Connection conn) {
    this.conn = conn;
}

/**
 * setChartId
 * @param chartId
 *
 */
public void setChartId(int chartId) {
    this.chartId = chartId;
}
```

```

}

/**
 * setChart
 * @param reportChart
 */
public void setReportChart(ReportChart reportChart) {
    this.reportChart = reportChart;
}

/**
 * setPresentationChart
 * @param presentationChart
 */
public void setReportChart(Char chart) throws Exception {
    this.reportChart.setChart(chart);
}

/**
 * getAlias()
 * @param table
 * @return alias
 */
private String getAlias(String table) {
    String alias;

    if (table.equalsIgnoreCase("PROBLEM") || table.equalsIgnoreCase("ITEM"))
alias = "p";
    else if (table.startsWith("PROBLEM_MODULE") ||
table.startsWith("ITEM_MODULE")) alias = "im";
    else if (table.startsWith("PROBLEM_RELEASE") ||
table.startsWith("ITEM_RELEASE")) alias = "ir";
    else if (table.startsWith("ATTACHMENT")) alias = "a";
    else if (table.startsWith("RELATIONSHIP_GROUP_")) alias = "rg";
    else alias = table;
    return alias;
}

/**
 * delete
 * @param conn
 * @param userId
 */
public void delete(Connection conn, String userId) throws Exception {
    this.conn = conn;
    if (this.newData == false) {
        this.deletedData = true;
        this.executeTransaction(userId);
    }
}

private String buildLookupsSql(String ddeName,
                                String ddeTable,
                                String ddeField,
                                String ddeDriverKey,
                                String ddeChildKey,
                                String ddeParentTable,
                                String ddeParentKey,
                                String ddeGrandParentTable,
                                String ddeGrandParentKey,
                                String ddeGrandChildKey,
                                String ddeLookupTable,

```

```

                                Report.java
                                String ddeLookupColumn1,
                                String ddeLookupColumn2,
                                String ddeLookupColumn3,
                                String ddeLookupKey,
                                String ddeDisplayType,
                                String ddeMultipleValue) throws Exception {

    String lookupSql = "";
    String temp = "";
    String tempwhere = "";

    try {
        dataAlias.clear();

// USED IN SELECT CLAUSE
/*-----*/
        if (ddeTable == null) ddeTable = "";
        if (ddeParentTable == null) ddeParentTable = "";

// is a field
        if (TextManager.isStringInvisible(ddeLookupKey)
            && TextManager.isStringInvisible(ddeLookupColumn1)) {
            lookupSql = getAlias(ddeTable) + "." + ddeField + " " + q + ddeName
+ q;
            dataAlias.put(ddeName, ddeMultipleValue);
        }
        else {

/* *****SPECIAL CASE HARD CODED!!!!!! ----- TO DO: use product_release_id
-----*/

            if ("PRODUCT_RELEASE".equalsIgnoreCase(ddeLookupTable)) {
                temp = "PRODUCT_RELEASE.PRODUCT_NAME " + Z.dbms.ojequals() +
"PROBLEM_RELEASE.PRODUCT_NAME " +
                " and PRODUCT_RELEASE.RELEASE " + Z.dbms.ojequals() +
"PROBLEM_RELEASE.RELEASE_FOUND ";
            }
            else if (ddeLookupKey.toUpperCase().indexOf("PROJECT_ID")>-1){
                temp = "PROJECT.PROJECT_ID" + Z.dbms.ojequals() +
getAlias(ddeTable) + "." + ddeField +
                " and PROJECT.AREA_ID" + Z.dbms.ojequals() +
getAlias(ddeTable) + ".AREA_ID";
            }
            else{
                temp = ddeLookupTable+ "." + ddeLookupKey + Z.dbms.ojequals() +
getAlias(ddeTable) + "." + ddeField;
            }

// is a calculation
            if (TextManager.isStringInvisible(ddeLookupKey)) {
                lookupSql = ddeLookupColumn1 + " " + q + ddeName + q;
                dataAlias.put(ddeName, ddeMultipleValue);
            }

// turns out that this is a normal field that is also used as a lookup for allowed
values
// here the table is null, but the key and the column are not
            else if (TextManager.isStringInvisible(ddeLookupTable)
                || ddeTable.equalsIgnoreCase(ddeLookupTable)) {
                lookupSql = getAlias(ddeTable) + "." + ddeLookupColumn1 + " " +

```

Report.java

```

q + ddeName + q;
dataAlias.put(ddeName, ddeMultipleValue);
if (TextManager.isStringVisible(ddeLookupColumn2)) {
    lookupSql += ", " + getAlias(ddeTable) + "." +
ddeLookupColumn2 +
        " " + q + ddeName + this.suffix1 + q;
    dataAlias.put(ddeName + this.suffix1, ddeMultipleValue);
}
if (TextManager.isStringVisible(ddeLookupColumn3)) {
    lookupSql += ", " + getAlias(ddeTable) + "." +
ddeLookupColumn2 +
        " " + q + ddeName + this.suffix2 + q;
    dataAlias.put(ddeName + this.suffix2, ddeMultipleValue);
}
if ("USER".equalsIgnoreCase(ddeDisplayType)) {
    lookupSql += ", " + getAlias(ddeTable) + "." + ddeField + "
" + ddeName + this.suffix3;
    dataAlias.put(ddeName + this.suffix3, ddeMultipleValue);
}
}

// lookup when problem, problem_release, or problem_module is the base table
else if ("ITEM".equalsIgnoreCase(ddeTable))
    || ddeTable.startsWith("PROBLEM_MODULE")
    || ddeTable.startsWith("PROBLEM_RELEASE")
    || ddeTable.startsWith("RELATIONSHIP")
    || ddeTable.startsWith("ITEM_MODULE")
    || ddeTable.startsWith("ITEM_RELEASE")
    || "ATTACHMENT".equalsIgnoreCase(ddeTable)) {
lookupSql = "(select distinct " + ddeLookupTable + "." +
ddeLookupColumn1 +
        " from " + ddeLookupTable + /*", " + ddeTable +*/
        " where " + temp + ")" + q + ddeName + q + " ";
dataAlias.put(ddeName, ddeMultipleValue);
if (TextManager.isStringVisible(ddeLookupColumn2)) {
    lookupSql += ", (select distinct " + ddeLookupTable + "." +
ddeLookupColumn2 +
        " from " + ddeLookupTable + /*", " + ddeTable
+*/
        " where " + temp + ")" + ddeName + this.suffix1
+ " ";
    dataAlias.put(ddeName + this.suffix1, ddeMultipleValue);
}
if (TextManager.isStringVisible(ddeLookupColumn3)) {
    lookupSql += ", (select distinct " + ddeLookupTable + "." +
ddeLookupColumn3 +
        " from " + ddeLookupTable + /*", " + ddeTable
+*/
        " where " + temp + ")" + ddeName + this.suffix2
+ " ";
    dataAlias.put(ddeName + this.suffix2, ddeMultipleValue);
}
if ("USER".equalsIgnoreCase(ddeDisplayType)) {
    lookupSql += ", " + getAlias(ddeTable) + "." + ddeField + "
" + ddeName + this.suffix3;
    dataAlias.put(ddeName + this.suffix3, ddeMultipleValue);
}
} // lookup when item_release, problem_module, problem is the parent
table
else if ("ITEM".equalsIgnoreCase(ddeParentTable)

```



```

Report.java
        ddeParentTable.startsWith("PROBLEM_MODULE")
        ddeParentTable.startsWith("PROBLEM_RELEASE")
        ddeParentTable.startsWith("ITEM_RELEASE")
        ddeParentTable.startsWith("ITEM_MODULE")
        "ATTACHMENT".equalsIgnoreCase(ddeParentTable)
"RELATIONSHIP_GROUP_PROBLEM".equalsIgnoreCase(ddeParentTable)
"RELATIONSHIP_GROUP_VIEW".equalsIgnoreCase(ddeParentTable)) {
        if ("".equals(ddeGrandParentTable)) {
            tempWhere = " where " /*+ ddeParentTable + "." +
ddeDriverKey + "=" +
getAlias(ddeParentTable) + "." + ddeDriverKey +
            " and " /*+ ddeTable + "." + ddeChildKey + Z.dbms.ojequals()
+
            getAlias(ddeParentTable) + "." + ddeParentKey +
            " and " + temp + ") ";
        } else {
            tempWhere = " where " + ddeParentTable + "." +
ddeGrandChildKey + Z.dbms.ojequals() +
            getAlias(ddeGrandParentTable) + "." +
ddeDriverKey +
            " and " + ddeTable + "." + ddeChildKey +
Z.dbms.ojequals() +
            ddeParentTable + "." + ddeParentKey +
            " and " + temp + ") ";
        }
        lookupSql = "(select distinct " + ddeLookupTable + "." +
ddeLookupColumn1 +
            " from " + ddeLookupTable + /*", " + ddeTable + ", "
+ ddeParentTable +*/
            tempWhere + q + ddeName + q + " ";
        dataAlias.put(ddeName, ddeMultipleValue);
        if (TextManager.isStringVisible(ddeLookupColumn2)) {
            lookupSql += ", (select distinct " + ddeLookupTable + "." +
ddeLookupColumn2 +
            " from " + ddeLookupTable + /*", " + ddeTable +
", " + ddeParentTable +*/
            tempWhere + ddeName + this.suffix1 + " ";
//Z.probe("RPT:MULTI2 alias2:" + ddeName + this.suffix1);
            dataAlias.put(ddeName + this.suffix1, ddeMultipleValue);
        }
        if (TextManager.isStringVisible(ddeLookupColumn3)) {
            lookupSql += ", (select distinct " + ddeLookupTable + "." +
ddeLookupColumn3 +
            " from " + ddeLookupTable + /*", " + ddeTable +
", " + ddeParentTable +*/
            tempWhere + ddeName + this.suffix2 + " ";
//Z.probe("RPT:MULTI2 alias2:" + ddeName + this.suffix2);
            dataAlias.put(ddeName + this.suffix2, ddeMultipleValue);
        }
        if ("USER".equalsIgnoreCase(ddeDisplayType)) {
            if ("".equals(ddeGrandParentTable))
                lookupSql += ", " + getAlias(ddeParentTable) + "." +
ddeField + " " + ddeName + this.suffix3;
            else
                lookupSql += ", " + ddeTable + "." + ddeField + " " +
ddeName + this.suffix3;

```

Report.java

```

        dataAlias.put(ddeName + this.suffix3, ddeMultipleValue);
    }
}
else {

/*
        lookupSql = "(select " + ddeLookupTable + "." +
ddeLookupColumn1 +
" from " + ddeLookupTable + ", " + ddeTable + ", " + ddeParentTable + ", " +
driverTable +
" where " + driverTable + "." + driverField + " = p.id " +
" and " + ddeTable + "." + ddeChildKey + Z.dbms.ojequals() + ddeParentTable + "." +
ddeParentKey +
" and " + ddeParentTable + "." + ddeParentKey +
Z.dbms.ojequals() + driverTable + "." + driverField +
" and " + ddeLookupTable + "." + ddeLookupKey +
Z.dbms.ojequals() + ddeTable + "." + ddeField + ")";
*/
//Z.probe("RPT:doing LOOKUP -- 2 tables out ----- ARGHH");

        dataAlias.put(ddeName, ddeMultipleValue);
    }
}
} catch (Exception e) {
    Z.log.writeToLog(Log.ERROR, this, "Error in method buildLookupSql:" +
e);
//ErrorWriter.write(e, ErrorWriter.LOG);
    throw e;
}
return lookupSql;
}

/**
 * processResults
 * @param ResultSet rs
 * @param ArrayList aliases
 * @return HashMap
 *
 * Processes the result set and groups information into lists if multiple
records are found
 */
private HashMap processResults(ResultSet rs,
                                HashMap aliases,
                                HashMap ddeDisplayedDates,
                                HashMap ddeTextFields,
                                HashMap ddeDisplayURLs,
                                HashMap dictionaryFields,
                                SesameSession session) throws Exception {

    boolean isList = false;
    String key = ""; // general key for hashmap lookups
    String pKey = ""; // the problem id as a key
    String listKey = ""; // temporary key value
    String multiple = "";
    String baseKey = "";
    String temp = "";
    ArrayList dataList = null;
    HashMap rows = null;
    HashMap fieldResults = null;
    HashMap recordResults = new HashMap();
    HashMap results = new HashMap();
    HashMap multiLookups = new HashMap();
    Set aliasSet = aliases.keySet();

```

Report.java

```

DDEntry ddeTemp = null;
DbTime dbt = null;
SecurityUser su = null;
String multi = "";
String multi1 = "";
String multi2 = "";
String multi3 = "";
ReportElement re = new ReportElement(null, null, isChart, hr);

try {
    dbt = new DbTime(session, conn);
    su = (SecurityUser) session.getAttribute("USER");

    while (rs.next()) {
        rows = new HashMap();

        pKey = rs.getString(PROBLEM_KEY_ALIAS); // this is the problem id

        multiLookups = new HashMap();

// Create a HashMap of ArrayLists for each problem ID to store field info
// -----
        if ((fieldResults = (HashMap) recordResults.get(pKey)) == null)
            fieldResults = new HashMap();

// Create the List if this is the first repeating value (id)
// -----
        Iterator aliasIterator = aliasSet.iterator();

        while (aliasIterator.hasNext()) {
            isList = false;
            key = (String) aliasIterator.next();
            multiple = (String) aliases.get(key);

            if (key.endsWith(sufx1) || key.endsWith(sufx2) ||
key.endsWith(sufx3))
                baseKey = key.substring(0, key.length() - sufx1.length());
            else
                baseKey = key;

            ddeTemp = (DDEntry) dictionaryFields.get(baseKey);

            if ("USER".equalsIgnoreCase(ddeTemp == null ? "" :
ddeTemp.getDisplayType())) {

// Check for a multi lookup --> currently ONLY when display_type = user
// --> TODO: display_as_url = y
// -----
//Z.probe("RPT:BASEKEY ____ ---- : " + baseKey);
//Z.probe("RPT:ddeTemp >>>" +baseKey);

                if (multiLookups.get(baseKey) == null) { // only process
unprocessed fields

                    multi1 = "";
                    multi2 = "";
                    multi3 = "";

// LOOKUP1

                    multi = rs.getString(baseKey);
                    multiLookups.put(baseKey, baseKey);

```

Report.java

```
// LOOKUP2
multi1 = rs.getString(baseKey+sufx1);
multiLookups.put(baseKey+sufx1,baseKey);

// LOOKUP3
multi2 = rs.getString(baseKey+sufx2);
multiLookups.put(baseKey+sufx2,baseKey);

// LOOKUP4
multi3 = rs.getString(baseKey+sufx3);
multiLookups.put(baseKey+sufx3,baseKey);

if ("Y".equalsIgnoreCase(multiple) &&
!key.equalsIgnoreCase(PROBLEM_KEY_ALIAS)){
    if ((dataList = (ArrayList) fieldResults.get(key))
== null) {
        dataList = new ArrayList();
    }
    isList = true;
    Z.log.writeToLog(Z.log.DEBUG5,"MULTIPLE >>>
this.getEmailLink = " +this.getEmailLink(multi3,multi,multi1,multi2));
    dataList.add(this.getEmailLink(multi3,
        multi,
        multi1,
        multi2));
    }
    else{
        Z.log.writeToLog(Z.log.DEBUG5,"SINGLE >>>
this.getEmailLink = " +this.getEmailLink(multi3,multi,multi1,multi2));
        fieldResults.put(baseKey, this.getEmailLink(multi3,
            multi,
            multi1,
            multi2));
    }
}
if (isList) fieldResults.put(baseKey, dataList);
}
else {

//if ("USER".equalsIgnoreCase(ddeTemp == null ? "" : ddeTemp.getDisplayType()))
//    multiLookups.put(key, baseKey);

// Get the result --- format based on display type
//-----
if (ddeDisplayedDates.get(key) != null)
    temp = formatDateString(dbt,
        su,
        Convert.toCalendar(rs.getTimestamp(key)),
        emptyStringSpacer);
else if (ddeDisplayURLs.get(key) != null)
    temp = getURLLink((String) ddeDisplayURLs.get(key),
        rs.getString(key));
else if (ddeTextFields.get(key) != null)
    temp = this.isTextOutputType()
        ? rs.getString(key)
        :
TextManager.convertUrl(getPrintableResult(rs.getString(key)));
```

```

                                Report.java
                                else
                                    temp = getPrintableResult(rs.getString(key));
                                // If there is a multiple value (don't want a list of the
problemKeys)
                                // get the dataList, add the value, put the dataList back in
//-----
                                if ("Y".equalsIgnoreCase(multiple) &&
!key.equalsIgnoreCase(PROBLEM_KEY_ALIAS)) {
                                    isList = true;
                                    if ((dataList = (ArrayList) fieldResults.get(key)) ==
null) {
                                        dataList = new ArrayList();
                                    }
                                    dataList.add(temp);
                                }
                                // This is a single value
                                //-----
                                else {
                                    fieldResults.put(key, temp);
                                    recordResults.put(pKey, fieldResults);
                                }
                                } //end if else user_type
                                if (isList) fieldResults.put(key, dataList);
                                } // finished processing fields
                                // Process the multi-row stuff here
                                // -----
                                if (isList) recordResults.put(pKey, fieldResults);
                                } // end while rs.next()
                                }
                                catch (Exception e) {
                                    Z.log.writeToLog(Log.ERROR, this, "Error in method process Results:" +
e);
                                //ErrorWriter.write(e, ErrorWriter.LOG);
                                    throw e;
                                }
                                }
                                ////Z.probe("RPT:HERE ARE THE RESUSLTS : " + recordResults);
                                return recordResults;
                                }

/**
 * getPrintableResult()
 * @param String s
 * @return String s
 *
 * utility function that either returns an empty string spacer or the value
 */
private String getPrintableResult(String temp) {
    String s = null;

    if (TextManager.isStringInvisible(temp))
        s = emptyStringSpacer;
    else {
        if (isTextOutputType())
            s = temp;
        else

```

```

Report.java
        s = TextManager.replaceLineFeed(TextManager.htmlEscape(temp));
    }
    return s;
}

/**
 * getFormattedNameForQuery()
 *
 * @param String fName
 * @param String lName
 * @return String
 *
 * Create the formatted name based on the Application Default value
 */
private String getFormattedNameForQuery(String tableAlias,
    String securityUserId,
    String firstName,
    String lastName) throws Exception {

    String display = Z.appDefaults.getAttribute("USERNAME_DISPLAY");
    String text = "";

    try {

        if ("FIRST".equalsIgnoreCase(display)) {
            text = (TextManager.isStringVisible(firstName)
                ? "(" + tableAlias + "." + firstName + " " + Z.dbms.cat() +
" ' ' " +
                Z.dbms.cat() + " " + tableAlias + "." + lastName + ")")
                : tableAlias + "." + lastName);
        }
        else if ("LAST".equalsIgnoreCase(display)) {
            text = (TextManager.isStringVisible(firstName)
                ? "(" + tableAlias + "." + lastName + " " + Z.dbms.cat() +
" ' , ' " +
                Z.dbms.cat() + " " + tableAlias + "." + firstName + ")")
                : tableAlias + "." + lastName);
        }
        else{
            text = tableAlias + "." + securityUserId;
        }
    } catch (Exception e) {
        Z.log.writeToLog(Log.ERROR, this, "Error in method
getFormattedNameForQuery:" + e);
        //ErrorWriter.write(e, ErrorWriter.LOG);
        throw e;
    }

    return text;
}

/**
 * getURLLink()
 *
 * @param String urlName
 * @param String displayName
 * @return String
 *
 * Create the email link based on the Application Default value
 */
private String getURLLink(String url, String value) throws Exception {

    StringBuffer html = new StringBuffer();
    Regex r = null;

```

Report.java

```

try {
    if (TextManager.isStringInvisible(value)) return
getPrintableResult(value);

    value = getPrintableResult(value);
    r = new Regex("(?e=~)~$~$~w+~$~$", value);

    if (this.isHTMLOutputType())
        html.append("<a href=\"\" + r.replaceAll(url) + \"\"><u>");

    html.append(value);

    if (this.isHTMLOutputType())
        html.append("</u></a>\n");
}
catch (Exception e) {
    Z.log.writeToLog(Log.ERROR, this, "Error in method getURLLink:" + e);
    throw e;
}
return html.toString();
}

/**
 * getEmailLink()
 * @param String ddName
 * @param String userId
 * @param String fName
 * @param String lName
 * @param String email
 * @return String
 *
 * Create the email link based on the Application Default value
 */
private String getEmailLink(String userId,
                             String firstName,
                             String lastName,
                             String email) throws Exception {

    StringBuffer html = new StringBuffer();

    try {

        html.append(SecurityUser.getUserNameDisplay(userId, firstName, lastName,
email, this.isHTMLOutputType()));

        if (html.length()==0) return emptyStringSpacer;
        else if (TextManager.isStringInvisible(html.toString())) return
emptyStringSpacer;
        else return html.toString();
    }
    catch (Exception e) {
        Z.log.writeToLog(Log.ERROR, this, "Error in method getEmailLink:" + e);
        throw e;
    }
}

private ArrayList buildReleaseInList(ArrayList sortedKeys,
String rQuery,
HashMap suppressedReleaseFilters) throws Exception{

    //Z.probe("IN buildReleaseInList ");

```


Report.java

```

StringBuffer query = new StringBuffer(rQuery);
ArrayList sortKeys = new ArrayList(sortedKeys);

try{
    String criteriaTableColumn = "";
    int criteriaSize = 0;
    Filter mf = null;
    ArrayList mfc = null;
    ArrayList rawfc = null;
    DDEntry dde = null;
    boolean skipComma = false;
    boolean hasNullCriteria = false;
    boolean wildcards = false;
    String rawValue = "";
    String processedValue = "";

    //
    // THIS IS A HACK
    // the right way to do this is to redo the release query using the
QueryBuilder getMasterList,
    // but change the driver table and driver talbe keys
    //
    // -- wont work with dates
    // -- doesn't care about dde's with parent table = "problem_release"
    //
    Z.probe("buildReleaseInList: SuppressedMultiReleaseFILTERS = " +
suppressedReleaseFilters );

    Iterator it = suppressedReleaseFilters.keySet().iterator();
    while(it.hasNext()){
        skipComma = false;
        mf = (Filter)suppressedReleaseFilters.get(it.next());
        rawfc = (ArrayList)mf.getFilterCriteriaResolved();
        mfc = new ArrayList();

        dde = mf.getDDEntry();

        hasNullCriteria = false;
        criteriaSize = rawfc.size();

        //
        // this is a hack
        //-----
        wildcards = false;
        for (int m = 0; m < criteriaSize; m++){
            rawValue = (String)rawfc.get(m);
            if ("{}null{}".equalsIgnoreCase(rawValue)) hasNullCriteria = true;

            processedValue =
TextManager.replace(rawValue,"*",Z.dbms.WILDCARD());

            if ("{}null{}".equalsIgnoreCase(processedValue)) hasNullCriteria =
true;

            if ("USER".equalsIgnoreCase(dde.getDisplayType()))
                mfc.add(processedValue.toUpperCase(Z.defaultLocale));
            else
                mfc.add(processedValue);
        }
    }
}

```

Report.java

```

    if (rawValue.indexOf("*") > -1) wildcards = true;
}

criteriaSize = mfc.size();

if (wildcards){
    Z.probe("buildReleaseInList: 30 ");

    //filter criteria loop
    for (int k = 0; k < criteriaSize; k++){
        if (k == 0) query.append(" and ( ");
        else query.append(" or ");
        query.append("UPPER("+ dde.getTableName() + "." +
dde.getColumnName() + ") ");
        query.append(" LIKE UPPER(?) ");
        sortKeys.add((String)mfc.get(k));
    }
    query.append(") ");

}
else{
    Z.probe("buildReleaseInList: 40 ");

    if (hasNullCriteria) query.append(" and ( ");
    else query.append(" and ");

    criteriaTableColumn = dde.getTableName() + "." +
dde.getColumnName();
    query.append(criteriaTableColumn + " in ( ");

    if (hasNullCriteria && criteriaSize == 1){
        query.append(" is null ");
    }

    for (int k = 0; k < criteriaSize; k++){
        if (hasNullCriteria){
            if ("{"null}".equalsIgnoreCase((String)mfc.get(k))){
                hasNullCriteria = true;
                skipComma = true;
                continue;
            }
        }
        if (k > 0 && !skipComma) query.append(", ");
        skipComma = false;
        query.append("?");

        sortKeys.add((String)mfc.get(k));
    }
    query.append(") "); // end in

    if (hasNullCriteria) query.append(" or " + criteriaTableColumn +
" is null )" ); // end or is null

    // query.append("ORDER BY DATE_CREATED DESC");
}
} // end if not wildcards
Z.probe("buildReleaseInList: 50 ");

```

```

    }
    catch(Exception e){
        throw e;
    }
    finally{
        Z.probe("@;^)->--<    the query is : " + query);
        return this.buildInList(sortedKeys, query.toString());
    }
}

/**
 * buildInList()
 * @param ArrayList sortedKeys
 * @param String query
 * Build the IN CLAUSE FOR THE RELEASE OR MODULE TABLES
 */
private ArrayList buildInList(ArrayList sortedKeys, String query) throws
Exception {

    ArrayList keys = new ArrayList();
    PreparedStatement stmt = null;
    ResultSet rs = null;

    try {
        Z.probe("RPT:  the in list Query is " + query);
        Z.probe("RPT:  the in list params are " + sortedKeys);

        stmt = conn.prepareStatement(query);

        // Bind the block of problem IDs
        //-----
        for (int k = 0; k < sortedKeys.size(); k++)
            stmt.setString(k + 1, (String) sortedKeys.get(k));

        rs = stmt.executeQuery();

        // Get the result set
        //-----
        while (rs.next()) {
            keys.add(rs.getString(1));
        }

    } catch (Exception e) {
        Z.log.writeToLog(Log.ERROR, this, "Error in method buileInList:" + e +
" with query: " + query);
        //ErrorWriter.write(e, ErrorWriter.LOG);
        throw e;
    } finally {
        if (stmt != null) stmt.close();
        return keys;
    }
}

/**
 * buildInList()
 * @param String query
 * Build the IN CLAUSE FOR THE RELEASE OR MODULE TABLES
 */
private StringBuffer buildInClause(ArrayList keys) throws Exception {

    StringBuffer inCriteria = new StringBuffer();

```

```

    try {
        inCriteria.append("(");
        for (int k = 0; k < keys.size(); k++) {
            if (k > 0) inCriteria.append(", ");
            inCriteria.append("?");
        }
        inCriteria.append(")");
    } catch (Exception e) {
        Z.log.writeToLog(Log.ERROR, this, "Error in method buildInClause:" +
e);
        throw e;
    }
    return inCriteria;
}

/**
 * formatDateString()
 * @param session
 * @param dbconn
 * @param dateStr
 * @return string
 */
private static String formatDateString(DbTime dbt,
    SecurityUser su,
    Calendar calendarDate,
    String emptyStringSpacer) throws Exception {

    String timestampString = emptyStringSpacer;

    if (calendarDate == null) return timestampString;

    try {
        dbt.setDbNow(calendarDate);
        timestampString = dbt.getFormattedDate(su.getDateFormat());
    } catch (Exception e) {
        throw e;
    } finally {
        return timestampString;
    }
}

/**
 * toLog()
 */
public void toLog() {
    if (this.fg != null) this.fg.toLog();
    if (this.l != null) this.l.toLog();
    if (this.so != null) this.so.toLog();
}

private String getHistoryEquivalent(String ddeTable) {

    if (("ITEM".equalsIgnoreCase(ddeTable)
        || "ITEM_MODULE".equalsIgnoreCase(ddeTable)
        || "ITEM_UDF".equalsIgnoreCase(ddeTable)
        || "ITEM_TEXT".equalsIgnoreCase(ddeTable)
        || "PROBLEM".equalsIgnoreCase(ddeTable)
        || "PROBLEM_RELEASE".equalsIgnoreCase(ddeTable)
        || "PROBLEM_MODULE".equalsIgnoreCase(ddeTable)
        || "PROBLEM_UDF".equalsIgnoreCase(ddeTable)
        || "PROBLEM_TEXT".equalsIgnoreCase(ddeTable))
        &&

```

```

Report.java
    TextManager.isStringVisible(ddeTable))
    ddeTable += "_HIST";
    return ddeTable;
}

private String getHistKey(String tableName, boolean isUDF) {
    if (tableName.startsWith("ITEM_UDF_HIST")
        || tableName.startsWith("ITEM_MODULE")
        || tableName.startsWith("PROBLEM_MODULE")
        || isUDF)
        return "LAST_DATE_UPDATED";
    else {
        return "TIMESTAMP";
    }
}

/**
 * destroy()
 * cleans up instance objects - result set & prepared statement
 */
public void destroy() throws Exception {
    try {
        dataAlias.clear();
    } catch (Exception e) {
        throw e;
    }
}
}
}

```

EXHIBIT D
22 PAGES

```

/**
 * $Workfile: Problem_UDF.java $
 *
 * $Revision: 86 $
 *
 * $Modtime: 12/30/02 12:54p $
 *
 *
 * Title:           Problem_UDF
 * Description:     UDF related Problem Functionality
 * Copyright:       Copyright (c) 2001
 * Company:         Sesame Technology
 * @author          Udele Malabanan
 * @version         1.0
 */

```

```
import java.text.*;
import java.util.*;
import java.sql.*;
import java.io.*;

import com.extraview.presentation.*;
import com.extraview.util.*;
import com.extraview.util.cmp.business.Misc;
import com.extraview.common.*;
import com.extraview.applogic.admin.*;
import com.extraview.applogic.*;
import com.extraview.applogic.history.*;
import com.extraview.applogic.security.*;
import com.extraview.dbms.dataMigration.*;
import com.sesame.log.*;

import com.stevesoft.pat.*;

import com.sesame.misc.*;
```

[illegible]

Page 1

```

                                Problem_UDF.java
// Sourcesfe Versioning Information
public static final String vssWorkfile = "$Workfile: Problem_UDF.java $";
public static final String vssRevision = "$Revision: 86 $";
public static final String vssModtime = "$Modtime: 12/30/02 12:54p $";

private String      problemId = null;
private String      udfId = null;
private String      value = null;
private String      valueNumber = null;
private float       valueAsNumber = 0;
private Calendar    valueDate = null;
private String      udfListId = null;
private Calendar    dateCreated = null;
private Calendar    lastDateUpdated = null;
private String      lastUpdatedByUser = null;
private String      createdByUser = null;

// from problem_text table
private String      idSeq = null;
private String      textSeq = null;
private String      userId = null;
private String      text = null;

private String      udfDisplayType = null;
private String      name = null; //ddName

private boolean     newData = false;
private boolean     modifiedData = false;
private boolean     deletedData = false;
private transient Connection dbconn = null;
private transient PrintWriter mOut = null;

// GETTERS
public String getProblemId() {
    return this.problemId;
}

public String getUdfId() {
    return this.udfId;
}

public String getValue() {
    return this.value;
}

public String getValueNumber() {
    return this.valueNumber;
}

public float getValueAsNumber() {
    return this.valueAsNumber;
}

public Calendar getValueDate() {
    return this.valueDate;
}

public String getUdfListId() {
    return this.udfListId;
}

public Calendar getDateCreated() {
    return this.dateCreated;
}

```


Problem_UDF.java

```

}

public Calendar getLastDateUpdated() {
    return this.lastDateUpdated;
}

public String getLastUpdatedByUser() {
    return this.lastUpdatedByUser;
}

public String getCreatedByUser() {
    return this.createdByUser;
}

public String getIdSeq() {
    return this.idSeq;
}

public String getTextSeq() {
    return this.textSeq;
}

/**
 * If this udf is a text_area, log_area or text_field, this method will
 * return a string representation of the text.
 */
public String getText() {
    return this.text;
}

public String getNonOverlappedText() {
    if (text != null && Integer.parseInt(this.textSeq) > 1) return
this.text.substring(Z.TEXT_OVERLAP);
    else return this.text;
}

public String getUserId() {
    return this.userId;
}

public String getUDFDisplayType() {
    return this.udfDisplayType;
}

public String getName() {
    if (name == null) Z.log.writeToLog(Z.log.ERROR, "CU: name is null on get");
    return this.name;
}

public boolean getModifiedData() {
    return this.modifiedData;
}

public boolean getNewData() {
    return this.newData;
}

public boolean getDeletedData() {
    return this.deletedData;
}

private Calendar timestamp = Calendar.getInstance(); //fn

```

```

Problem_UDF.java
public Calendar getTimestamp() {
    return this.timestamp;
}

// SETTERS
public void setProblemId(String problemId) {
    this.problemId = problemId;
}

public void setUdfId(String udfId) {
    this.udfId = udfId;
}

public void setValue(String value) {
    this.value = value;
}

public void setValueNumber(String valueNumber) {
    this.valueNumber = valueNumber;
    this.valueAsNumber = Float.parseFloat(valueNumber);
}

public void setValueAsNumber(float valueAsNumber) {
    this.valueAsNumber = valueAsNumber;
    this.valueNumber = String.valueOf(valueAsNumber);
}

public void setValueDate(Calendar valueDate) {
    this.valueDate = valueDate;
}

public void setUdfListId(String udfListId) {
    this.udfListId = udfListId;
}

public void setDateCreated(Calendar dateCreated) {
    this.dateCreated = dateCreated;
}

public void setLastDateUpdated(Calendar lastDateUpdated) {
    this.lastDateUpdated = lastDateUpdated;
}

public void setLastUpdatedByUser(String lastUpdatedByUser) {
    this.lastUpdatedByUser = lastUpdatedByUser;
}

public void setCreatedByUser(String createdByUser) {
    this.createdByUser = createdByUser;
}

public void setIdSeq(String idSeq) {
    this.idSeq = idSeq;
}

public void setTextSeq(String textSeq) {
    this.textSeq = textSeq;
}

public void setText(String text) {
    this.text = text;
}

```

```

                                Problem_UDF.java
public void setUDFDisplayType(String type) {
    this.udfDisplayType = type;
}

public void setName(String name) {
    if (name == null) Z.log.writeToLog(Z.log.ERROR, "PU: name is null");
    this.name = name;
}

public void setTimestamp(Calendar timestamp) {
    this.timestamp = timestamp;
}

/**
 * Sets the value of newData. When executeTransaction(String userId) for this
Problem_Release
 * object is executed, newData determines whether an insert is required. If
this is set to true
 * then this is a new record and it needs to be inserted into the database.
 * @param boolean val
 */
public void setModifiedData(boolean val) {
    this.modifiedData = val;
}

/**
 * Sets the value of newData. When executeTransaction(String userId) for this
Problem_UDF
 * object is executed, newData determines whether an insert is required. If
this is set to true
 * then this is a new record and it needs to be inserted into the database.
 * @param boolean val
 */
public void setNewData(boolean val) {
    this.newData = val;
}

/**
 * Sets the value of deletedData. When executeTransaction(String userId) for
this Problem_UDF
 * object is executed, deletedData determines if this record should be deleted. If
this is set to true
 * then this record should be deleted from the database.
 * @param boolean val
 */
public void setDeletedData(boolean val) {
    this.deletedData = val;
}

public Problem_UDF(String problemId, String pName) {
    this.problemId = problemId;
    this.name = pName;
}

/**
 * populateObj populates a Problem_UDF object with the data
 * in HashMap that corresponds to a row in the problem_udf view/
 * item_udf table
 */
public void populateObj(Map data) {
    this.udfId = (String) data.get("UDF_ID");
    this.value = (String) data.get("VALUE");
    setValueNumber((String) data.get("VALUE_NUMBER"));
}

```

```

        Problem_UDF.java
        if (TextManager.isStringVisible((String) data.get("VALUE_DATE"))) {
            this.valueDate = MetadataXMLUpdater.getCalendar((String)
data.get("VALUE_DATE"));
        }
        this.udfListId = (String) data.get("UDF_LIST_ID");
        if (TextManager.isStringVisible((String) data.get("DATE_CREATED"))) {
            this.dateCreated = MetadataXMLUpdater.getCalendar((String)
data.get("DATE_CREATED"));
        }
        if (TextManager.isStringVisible((String) data.get("LAST_DATE_UPDATED"))) {
            this.lastDateUpdated = MetadataXMLUpdater.getCalendar((String)
data.get("LAST_DATE_UPDATED"));
        }
        this.lastUpdatedByUser = (String) data.get("LAST_UPDATED_BY_USER");
        this.createdByUser = (String) data.get("CREATED_BY_USER");
        this.problemId = (String) data.get("ITEM_ID");

        Z.log.writeToLog(Z.log.DEBUG, "populateObj ProblemUdf udfId " + udfId);
        Z.log.writeToLog(Z.log.DEBUG, "populateObj ProblemUdf value " + value);
        Z.log.writeToLog(Z.log.DEBUG, "populateObj ProblemUdf valueNumber " +
valueNumber);
        Z.log.writeToLog(Z.log.DEBUG, "populateObj ProblemUdf valueDate " +
valueDate);
        Z.log.writeToLog(Z.log.DEBUG, "populateObj ProblemUdf udfListId " +
udfListId);
        Z.log.writeToLog(Z.log.DEBUG, "populateObj ProblemUdf dateCreated " +
dateCreated);
        Z.log.writeToLog(Z.log.DEBUG, "populateObj ProblemUdf lastDateUpdated " +
lastDateUpdated);
        Z.log.writeToLog(Z.log.DEBUG, "populateObj ProblemUdf lastUpdatedByUser " +
lastUpdatedByUser);
        Z.log.writeToLog(Z.log.DEBUG, "populateObj ProblemUdf createdByUser " +
createdByUser);
        Z.log.writeToLog(Z.log.DEBUG, "populateObj ProblemUdf problemId " +
problemId);
    }

    public Problem_UDF(Problem_UDF pUDF) {
        this.problemId = pUDF.problemId;
        this.udfId = pUDF.udfId;
        this.value = pUDF.value;
        this.valueNumber = pUDF.valueNumber;
        this.valueAsNumber = pUDF.valueAsNumber;
        this.valueDate = pUDF.valueDate;
        this.udfListId = pUDF.udfListId;
        this.dateCreated = pUDF.dateCreated;
        this.lastDateUpdated = pUDF.lastDateUpdated;
        this.lastUpdatedByUser = pUDF.lastUpdatedByUser;
        this.createdByUser = pUDF.createdByUser;

        this.udfDisplayType = pUDF.udfDisplayType;

        this.idSeq = pUDF.idSeq;
        this.textSeq = pUDF.textSeq;
        this.text = pUDF.text;
        this.name = pUDF.name;
    }

    public Problem_UDF getClone() {
        Problem_UDF clone = null;

        try {

```

```

        Problem_UDF.java
        clone = (Problem_UDF) super.clone();
    } catch (Exception e) {; // pass
    } finally {
        return clone;
    }
}

/**
 * <p> Sets udf text values in hashmap. This is only used by presentation.
 * It sets 2 items in the HashMap. The first item is the 'TEXT' value in the
problem_text table
 * for udf_id assoicated with this problem id. The second is the value from
the
 * problem_udf table that is associated with this problem id. This is either
the
 * value, value_date, value_number or udf_list_id depending on udf_type.</p>
 * @param String problemId
 * @param Connection dbconn
 * @param HashMap hash
 */
public static void setHash(SesameSession session, Connection dbconn, String
problemId, HashMap hash,
    boolean includeLogAreas)
    throws Exception {

    String udfType = null;
    String prevUdfId = null;
    String udfId = null;
    String udfName = null;

    String text = null;

    StringBuffer udfText = new StringBuffer();

    if (!includeLogAreas) {
        udfText.append(" select pt.udf_id, id_seq, text_seq, text, u.name ");
        udfText.append(" from udf u, data_dictionary dd, problem_text pt ");
        udfText.append(" where pt.problem_id = ? ");
        udfText.append(" and u.udf_id = pt.udf_id ");
        udfText.append(" and dd.name = u.name ");
        udfText.append(" and dd.display_type <> 'LOGAREA' ");
        udfText.append(" order by 1, 2, 3 ");
    } else {
        udfText.append(" select pt.udf_id, id_seq, text_seq, text, u.name from
problem_text pt, udf u where problem_id = ? ");
        udfText.append(" and u.udf_id = pt.udf_id ");
        udfText.append(" order by 1, 2, 3 ");
    }
    StringBuffer udfVal = new StringBuffer(" select pu.UDF_ID, VALUE,
VALUE_NUMBER, VALUE_DATE, pu.UDF_LIST_ID, TITLE, u.name ");

    udfVal.append(" from udf_list ul, problem_udf pu, udf u ");
    udfVal.append(" where problem_id = ? ");
    udfVal.append(" and ul.udf_list_id " + Z.dbms.ojequals() + "
pu.udf_list_id");
    udfVal.append(" and pu.udf_id = u.udf_id ");

    PreparedStatement statement = dbconn.prepareStatement(udfText.toString());

    statement.setString(1, problemId);
    try{
        // get long text udfs

```

```

        Problem_UDF.java
        ResultSet result = statement.executeQuery();

        StringBuffer sb = null;
        while ( result.next() ) {
            udfId = result.getString("UDF_ID");
            udfName = result.getString("NAME");
            if(result.getInt("TEXT_SEQ") > 1){
                sb.setLength(sb.length() - Z.TEXT_OVERLAP);
                sb.append(result.getString("TEXT"));
            } else {
                sb = new StringBuffer(result.getString("TEXT"));
            }

            hash.put(udfName, sb.toString());
        }
    } catch (Exception e) {
        ErrorWriter.write(e, ErrorWriter.LOG);
    } finally {
        statement.close();
    }

    PreparedStatement statement2 = dbconn.prepareStatement(udfVal.toString());

    statement2.setString(1, problemId);
    // get rest of udfs
    try {
        ResultSet result2 = statement2.executeQuery();

        while (result2.next()) {
            udfId = result2.getString("UDF_ID");
            udfName = result2.getString("NAME");
            udfType = Z.dictionary.getDisplayType(dbconn, udfName);
            String value = null;

            // get the value from the correct column, depending on the udf type
            if (DataDictionary.isListType(udfType)) {
                value = result2.getString("UDF_LIST_ID");
            } else if (udfType.equals("DATE")) {
                Timestamp dateVal = result2.getTimestamp("VALUE_DATE");
                Calendar date = Convert.toCalendar(dateVal);

                DbTime dbt = new DbTime(session, dbconn);

                if (date != null) {
                    dbt.setDbNow(date);
                    value = dbt.getShortDate();
                }
            } else if (udfType.equals("NUMBER")) {
                value = result2.getString("VALUE_NUMBER");
            } else {
                value = result2.getString("VALUE");
            }

            if (hash.get(udfName) != null) { // for multi-valued udfs, add to
the array
                String[] udfList = null;

                if (hash.get(udfName) instanceof String[]) {
                    String[] currlist = (String[]) hash.get(udfName);

```

Problem_UDF.java

```
        udflist = new String[currlist.length + 1];
        for (int i = 0; i < currlist.length; i++) {
            udflist[i] = currlist[i];
        }

        udflist[currlist.length] = value;
    } else {
        udflist = new String[2];
        udflist[0] = (String) hash.get(udfName);
        udflist[1] = value;
    }

    hash.put(udfName, udflist);
} else { // not multi-valued, just put in hashmap
    hash.put(udfName, value);

    if ("DATE".equals(udfType)) {
        hash.put((udfName + "_EVDISPLAY"), value);
    } else {
        hash.put((udfName + "_EVDISPLAY"),
result2.getString("TITLE"));
    }
}

}

} catch (Exception e) {
    ErrorWriter.write(e, ErrorWriter.LOG);
} finally {
    statement2.close();
}
}

/**
 * <p> Sets the connection for the object </p>
 * @param dbconn
 */
public void setConnection(Connection dbconn) {
    this.dbconn = dbconn;
}

/**
 * <p> Sets the PrintWriter for the object </p>
 * @param writer
 */
public void setPrintWriter(PrintWriter writer) {
    this.mOut = writer;
}

/**
 * <p> Checks itself for its display type and sets the appropriate column
value</p>
 */
public void validate(Connection dbconn, SesameSession session) throws
ValidationException, Exception{

    SesameMessageFormat smf = new SesameMessageFormat(dbconn, session);
```


Problem_UDF.java

```
if ("NUMBER".equals(udfDisplayType)) {
    try {
        Float v = Float.valueOf(this.value);
    } catch (NumberFormatException e) {
        smf.clear();
        smf.appendString(this.value);
        smf.append(" is not a valid number.");

        throw new ValidationException(smf.getFormattedMessage());
    }
    setValueNumber(this.value);
    this.value = null;
} else if ("DATE".equals(udfDisplayType)) {
    java.util.Date date = null;

    try{
        date = Convert.getDateFromString(session, this.value);
        if (date == null) throw new ValidationException();
    } catch (ValidationException ve){
        smf.clear();
        smf.appendString(this.value);
        smf.append(" is not a valid date.");

        throw new ValidationException (smf.getFormattedMessage());
    } catch (Exception e){
        smf.clear();
        smf.appendString(this.value);
        smf.append(" is not a valid date.");

        throw new ValidationException (smf.getFormattedMessage());
    }

    Calendar cal = Calendar.getInstance(session.getLocale());

    cal.setTime(date);
    setValueDate(cal);
    this.value = null;
} else if ("USER".equals(udfDisplayType)){
    if (!Misc.isAffirmative(Z.appDefaults.getAttribute("SSO_STATE")) &&
        TextManager.isStringVisible(this.value) &&
        !SecurityUser.doesUserIdExist(dbconn, this.value.toUpperCase())){
        smf.clear();
        smf.appendString(this.value);
        smf.append(" is not a valid user.");

        throw new ValidationException (smf.getFormattedMessage());
    }
}

} else if (DataDictionary.isListType(udfDisplayType)) {
    this.udfListId = this.value;
    this.value = null;
} else if (DataDictionary.isProblemTextType(udfDisplayType)) {
    this.text = this.value;
```

```

        this.value = null;
    }

}

/**
 * This method returns the udfId for a specific udf.
 * @param Connection conn
 * @param String name - the name of the udf.
 */
// this should eventually be moved into udf class
public static String getUdfId(Connection conn, String name) throws Exception {
    String id = "";
    String sql = "select udf_id from udf where name = upper(?) ";
    PreparedStatement statement = null;

    try {
        statement = conn.prepareStatement(sql);
        statement.setString(1, name);
        ResultSet result = statement.executeQuery();

        while (result.next()) {
            id = result.getString(1);
        }
    } catch (Exception e) {
        return "error retrieving id; " + e;
    } finally {
        statement.close();
    }
    return id;
}

/**
 * Returns the name of the udf for the specific udfId.
 * @param Connection conn
 * @param String udfId
 */
public static String getName(Connection conn, String udfId) throws Exception {
    String name = "";
    String sql = "select name from udf where udf_id = ? ";
    PreparedStatement statement = null;

    try {
        statement = conn.prepareStatement(sql);
        statement.setString(1, udfId);
        ResultSet result = statement.executeQuery();

        while (result.next()) {
            name = result.getString(1);
        }
    } catch (Exception e) {
        ErrorWriter.write(e, ErrorWriter.LOG);
    } finally {
        statement.close();
    }
    return name;
}

/**
 * Returns a representation of PROBLEM_UDF and PROBLEM_TEXT tables for the
 * associated problemId.
 * @param Connection dbconn

```

Problem_UDF.java

* @param String problemId
 * @return ArrayList of 2 Problem_UDF objects. The first represents the values
 in the
 * PROBLEM_UDF table and the second represents the values in the PROBLEM_TEXT.
 */

```
public static ArrayList getReference(Connection dbconn, String problemId) {
```

```
    Problem_UDF udf = null;  
    ArrayList udfList = new ArrayList();
```

```
    PreparedStatement ps1 = null;  
    PreparedStatement ps2 = null;
```

```
    StringBuffer sql1 = new StringBuffer();
```

```
    sql1.append(" SELECT PU.UDF_ID UDF,");  
    sql1.append("    PU.VALUE,");  
    sql1.append("    PU.VALUE_NUMBER VNUM,");  
    sql1.append("    PU.VALUE_DATE VDATE,");  
    sql1.append("    PU.UDF_LIST_ID VUDF,");  
    sql1.append("    PU.DATE_CREATED CREATED,");  
    sql1.append("    PU.LAST_DATE_UPDATED UPDATED,");  
    sql1.append("    PU.LAST_UPDATED_BY_USER UPDATED_BY,");  
    sql1.append("    PU.CREATED_BY_USER CREATED_BY,");  
    sql1.append("    U.NAME");  
    //sql1.append("    DD.DISPLAY_TYPE");  
    sql1.append(" FROM");  
    sql1.append("    UDF U,");  
    sql1.append("    PROBLEM_UDF PU");  
    sql1.append(" WHERE PROBLEM_ID = ?");  
    sql1.append(" AND U.UDF_ID = PU.UDF_ID");  
    //sql1.append(" AND DD.NAME = U.NAME");  
    sql1.append(" order by pu.last_date_updated desc");
```

```
    StringBuffer sql2 = new StringBuffer();
```

```
    sql2.append(" SELECT PT.UDF_ID UDF,");  
    sql2.append("    PT.ID_SEQ,");  
    sql2.append("    PT.TEXT_SEQ,");  
    sql2.append("    PT.USER_ID,");  
    sql2.append("    PT.TIMESTAMP,");  
    sql2.append("    PT.TEXT,");  
    sql2.append("    U.NAME");  
    //sql2.append("    DD.DISPLAY_TYPE");  
    sql2.append(" FROM");  
    sql2.append("    UDF U,");  
    sql2.append("    PROBLEM_TEXT PT");  
    sql2.append(" WHERE PT.PROBLEM_ID = ?");  
    sql2.append(" AND U.UDF_ID = PT.UDF_ID");  
    //sql2.append(" AND DD.NAME = U.NAME");  
    sql2.append(" order by pt.timestamp desc, pt.udf_id, pt.text_seq ");
```

```
    try {  
        ps1 = dbconn.prepareStatement(sql1.toString());  
        ps1.setString(1, problemId);  
  
        ResultSet rs1 = ps1.executeQuery();  
  
        while (rs1.next()) {  
            udf = new Problem_UDF(problemId, rs1.getString("NAME"));  
            udf.newData = false;  
            udf.udfId = rs1.getString("UDF");  
            udf.value = rs1.getString("VALUE");
```

```

        Problem_UDF.java
        udf.setValueAsNumber(rs1.getFloat("VNUM"));
        udf.valueDate = Convert.toCalendar(rs1.getTimestamp("VDATE"));
        udf.udfListId = rs1.getString("VUDF");
        udf.dateCreated = Convert.toCalendar(rs1.getTimestamp("CREATED"));
        udf.lastDateUpdated =
Convert.toCalendar(rs1.getTimestamp("UPDATED"));
        udf.setLastUpdatedByUser(rs1.getString("UPDATED_BY"));
        udf.setCreatedByUser(rs1.getString("CREATED_BY"));
        DDEntry dde = DataDictionary.getDDEntry(dbconn, udf.getName());
        udf.setUDFDisplayType(dde.getDisplayType());
        udfList.add(udf);
    }

    //close the first set of cursors
    ps1.close();
    rs1.close();

    ps2 = dbconn.prepareStatement(sql2.toString());
    ps2.setString(1, problemId);
    ResultSet rs2 = ps2.executeQuery();

    while (rs2.next()) {
        udf = new Problem_UDF(problemId, rs2.getString("NAME"));
        udf.newData = false;
        udf.udfId = rs2.getString("UDF");
        udf.idSeq = rs2.getString("ID_SEQ");
        udf.textSeq = rs2.getString("TEXT_SEQ");
        udf.userId = rs2.getString("USER_ID");
        udf.lastDateUpdated =
Convert.toCalendar(rs2.getTimestamp("TIMESTAMP"));
        udf.text = rs2.getString("TEXT");
        DDEntry dde = DataDictionary.getDDEntry(dbconn, udf.getName());
        udf.setUDFDisplayType(dde.getDisplayType());
        udfList.add(udf);
    }
    ps2.close();
    rs2.close();
} catch (Exception e) {
    ErrorWriter.write(e, ErrorWriter.LOGERR);
} finally {
    try {
        if (ps1 != null) ps1.close();
        if (ps2 != null) ps2.close();
    } catch (Exception e) {} // pass
}
}
return udfList;
}

/**
 * This method gets the value for a udf of type list for a specific problemId
 * and udfId. This method does involve a trip to the DB to get this value.
 */
public String getUdfListValue(Connection dbcon, String problemId, String udfId)
{
    String sqlstr = null;
    PreparedStatement ps = null;
    ResultSet rs = null;
    String value = null;

    sqlstr = " select ul.title title " +
        " from udf_list ul, " +
        " problem_udf pu " +

```

```

                                Problem_UDF.java
        "   where pu.problem_id = ? " +
        "   and   pu.udf_id = ? " +
        "   and   pu.udf_list_id = ul.udf_list_id ";

    try {
        //precompile the ps
        ps = dbcon.prepareStatement(sqlstr);
        //set the placeholder
        ps.setString(1, problemId);
        ps.setString(2, udfId);
        //execute the ps
        rs = ps.executeQuery();
        //get the result and check where to put it
        while (rs.next()) {
            value = rs.getString("title");
        }
    } catch (Exception e) {
        ErrorWriter.write(e, ErrorWriter.LOGERR);
    } finally {
        try {
            if (ps != null) ps.close();
            if (rs != null) rs.close();
        } catch (Exception e) {
            ErrorWriter.write(e, ErrorWriter.LOGERR);
        }
    }
    //end try
    return value;
} //end method.

```

```

/**
 * this is for getting multi-valued udf list values
 */

```

```

public static ArrayList getUdfListValues(Connection dbcon, String problemId,
String udfId) {
    String sqlstr = null;
    PreparedStatement ps = null;
    ResultSet rs = null;
    ArrayList valueList = new ArrayList();
    String value = null;

    sqlstr = " select ul.title title " +
        "   from udf_list ul, " +
        "   problem_udf pu " +
        "   where pu.problem_id = ? " +
        "   and   pu.udf_id = ? " +
        "   and   pu.udf_list_id = ul.udf_list_id " +
        "   order by ul.title ";

    try {
        //precompile the ps
        ps = dbcon.prepareStatement(sqlstr);
        //set the placeholder
        ps.setString(1, problemId);
        ps.setString(2, udfId);
        //execute the ps
        rs = ps.executeQuery();
        //get the result and check where to put it
        while (rs.next()) {
            value = rs.getString("title");
            valueList.add(value);
        }
    } catch (Exception e) {
        ErrorWriter.write(e, ErrorWriter.LOGERR);
    }
}

```

Problem_UDF.java

```

    } finally {
        try {
            if (ps != null) ps.close();
            if (rs != null) rs.close();
        } catch (Exception e) {
            ErrorWriter.write(e, ErrorWriter.LOGERR);
        }
    } //end try
    //Z.probe("ProblemUdf getUdfListValues valueList " + valueList);
    return valueList;
} //end method.

/**
 * getLogTimestamp gets the problem text timestamp for log
 */
public static Calendar getLogTimestamp(Connection dbconn, String ddName, String
problemId) {
    String sqlstr = null;
    PreparedStatement ps = null;
    ResultSet rs = null;
    Calendar timestamp = null;

    sqlstr = " select pt.timestamp " +
        " from problem_text pt, udf u " +
        " where u.udf_Id = pt.udf_Id " +
        " and u.name = ? " +
        " and pt.problem_id = ? ";

    try {
        //precompile the ps
        ps = dbconn.prepareStatement(sqlstr);
        //set the placeholder
        ps.setString(1, ddName);
        ps.setString(2, problemId);
        //execute the ps
        rs = ps.executeQuery();
        //get the result and check where to put it
        while (rs.next()) {
            timestamp = Convert.toCalendar(rs.getTimestamp("TIMESTAMP"));
        }
    } catch (Exception e) {
        ErrorWriter.write(e, ErrorWriter.LOGERR);
    } finally {
        try {
            if (ps != null) ps.close();
            if (rs != null) rs.close();
        } catch (Exception e) {
            ErrorWriter.write(e, ErrorWriter.LOGERR);
        }
    } //end try

    return timestamp;
} //end method.

/**
 * <p> Transactional interface </p>
 * @param problem_id
 */
public void executeTransaction(String userId) throws Exception {
    Z.log.writeToLog(Z.log.DEBUG1, "Entering Problem_UDF.executeTransaction");
    StringBuffer sql = new StringBuffer();
    String SYSDATE = Z.dbms.CURRENT_DATE();

```

```

                                Problem_UDF.java
java.sql.Timestamp timestamp = Convert.toDate(this.timestamp);
PreparedStatement statement = null;
String udfType = DataDictionary.getDisplayType(dbconn, name);

//error message
StringBuffer outErrMsg = new StringBuffer();

outErrMsg.append("<SCRIPT>");
outErrMsg.append("alert(\"There has been an error in adding the data to the
database. Please contact the system administrator.\");");
outErrMsg.append("</SCRIPT>");

// make sure there is a transaction to execute
if (!newData && !deletedData && !modifiedData) {
    Z.log.writeToLog(Z.log.DEBUG1,"no directions");
    return;
}

// no database record exists, so do nothing
if (deletedData && newData) {
    Z.log.writeToLog(Z.log.DEBUG1,"both deleted data and new data");
    newData = false;
    deletedData = false;
    modifiedData = false;
    return;
}

if ("USER".equals(udfType) && this.value != null) this.value =
this.value.toUpperCase();

try {
    if (deletedData) {
        Z.log.writeToLog(Z.log.DEBUG1,"deleting udf id " + this.udfId + "
from problem " + this.problemId);
        if (DataDictionary.isProblemTextType(udfDisplayType)) {
            sql.append(" delete from problem_text ");
            sql.append(" where problem_id = ? ");
            sql.append(" and udf_id = ? ");

            statement = dbconn.prepareStatement(sql.toString());
            statement.setString(1, this.problemId);
            statement.setString(2, this.udfId);
            statement.executeUpdate();
            //update problemTextHist
            ProblemUdfHist.updateProblemTextHist(dbconn, timestamp,
problemId);
        } else {
            sql.append(" delete from problem_udf ");
            sql.append(" where problem_id = ? ");
            sql.append(" and udf_id = ? ");

            statement = dbconn.prepareStatement(sql.toString());
            statement.setString(1, this.problemId);
            statement.setString(2, this.udfId);
            statement.executeUpdate();
            //update problemUdfHist timestamp
            // ProblemUdfHist.updateProblemUdfHist(dbconn, timestamp,
problemId);
        }
    } else if (newData) {
        Z.log.writeToLog(Z.log.DEBUG1,"inserting udf id " + this.udfId + "

```



```

for problem " + this.problemId);
    if (DataDictionary.isProblemTextType(udfDisplayType)) {
        sql.append(" insert into PROBLEM_TEXT ");
        sql.append(" (PROBLEM_ID,");
        sql.append(" UDF_ID,");
        sql.append(" ID_SEQ,");
        sql.append(" TEXT_SEQ,");
        sql.append(" USER_ID,");
        sql.append(" TIMESTAMP,");
        sql.append(" TEXT)");
        sql.append(" values(?,?,?,?,?,?,?)");

        statement = dbconn.prepareStatement(sql.toString());
        statement.setString(1, this.problemId);
        statement.setString(2, this.udfId);
        statement.setString(3, this.idSeq);
        statement.setString(4, this.textSeq);
        statement.setString(5, userId);
        statement.setTimestamp(6, timestamp);
        statement.setString(7, this.text);
        statement.executeUpdate();
    } else {
        sql.append(" insert into PROBLEM_UDF ");
        sql.append(" (PROBLEM_ID,");
        sql.append(" UDF_ID,");
        sql.append(" VALUE,");
        sql.append(" VALUE_NUMBER,");
        sql.append(" VALUE_DATE,");
        sql.append(" UDF_LIST_ID,");
        sql.append(" DATE_CREATED,");
        sql.append(" LAST_DATE_UPDATED,");
        sql.append(" LAST_UPDATED_BY_USER,");
        sql.append(" CREATED_BY_USER)");
        sql.append(" values(?,?,?,?,?,?,?,?,?,?) ");
        // should evaluate data_type before setting variable
        // this will eliminate the need for complicated execute routines
        statement = dbconn.prepareStatement(sql.toString());
        statement.setString(1, this.problemId);
        statement.setString(2, this.udfId);
        statement.setString(3, this.value);
        statement.setFloat(4, this.valueAsNumber);
        statement.setTimestamp(5, (valueDate != null) ?
Convert.toDate(this.valueDate) : null);
        statement.setString(6, this.udfListId);
        statement.setTimestamp(7, (this.dateCreated == null ? timestamp
: Convert.toDate(dateCreated)));
        statement.setTimestamp(8, (this.lastDateUpdated == null ?
timestamp : Convert.toDate(this.lastDateUpdated)));
        statement.setString(9, (this.lastUpdatedByUser == null ? userId
: this.lastUpdatedByUser));
        statement.setString(10, (this.createdByUser == null ? userId :
this.createdByUser));
        //statement.setString(9, userId);
        //statement.setString(10, userId);

        statement.executeUpdate();
    }
} else if (modifiedData) {
    Z.log.writeToLog(Z.log.DEBUG1,"updating " + this.udfId + " for
problem " + this.problemId);
    if ("TEXTAREA".equals(udfDisplayType) ||

```

```

Problem_UDF.java
"PRINTTEXT".equals(udfDisplayType)) {

// if (this.textSeq.equals(new Integer(1) )) { how
did this ever work ???? - UM
// why are we just deleting the first one? - UM
if (this.textSeq.equals("1")) {
    sql.append(" delete from problem_text ");
    sql.append(" where problem_id = ? ");
    sql.append(" and udf_id = ? ");

    Z.log.writeToLog(Z.log.DEBUG1,"<===problem udf deletesql==>
" + sql.toString());

    statement = dbconn.prepareStatement(sql.toString());
    statement.setString(1, this.problemId);
    statement.setString(2, this.udfId);
    statement.executeUpdate();
    //update problemTextHist
    ProblemUdfHist.updateProblemTextHist(dbconn, timestamp,
problemId);
}

sql.setLength(0);
sql.append(" insert into PROBLEM_TEXT ");
sql.append(" (PROBLEM_ID, ");
sql.append(" UDF_ID, ");
sql.append(" ID_SEQ, ");
sql.append(" TEXT_SEQ, ");
sql.append(" USER_ID, ");
sql.append(" TIMESTAMP, ");
sql.append(" TEXT) ");
sql.append(" values(?,?,?,?,?,?,?)");

Z.log.writeToLog(Z.log.DEBUG1,"<===problem udf insertsql==> " +
sql.toString());

statement = dbconn.prepareStatement(sql.toString());
statement.setString(1, this.problemId);
statement.setString(2, this.udfId);
statement.setString(3, this.idSeq);
statement.setString(4, this.textSeq);
statement.setString(5, userId);
statement.setTimestamp(6, timestamp);
statement.setString(7, this.text);
statement.executeUpdate();
} else {

    sql.append(" UPDATE PROBLEM_UDF ");
    sql.append(" SET VALUE ");
    sql.append(" VALUE_NUMBER ");
    sql.append(" VALUE_DATE ");
    sql.append(" UDF_LIST_ID ");
    sql.append(" LAST_DATE_UPDATED ");
    sql.append(" LAST_UPDATED_BY_USER ");
    sql.append(" WHERE PROBLEM_ID ");
    sql.append(" AND UDF_ID ");

    statement = dbconn.prepareStatement(sql.toString());
    statement.setString(1, this.value);
    statement.setFloat(2, this.valueAsNumber);
    statement.setTimestamp(3, (valueDate != null) ?
Convert.toDate(this.valueDate) : null);
    statement.setString(4, this.udfListId);

```

```

        Problem_UDF.java
        statement.setTimestamp(5, (this.lastDateUpdated == null ?
timestamp : Convert.toDate(this.lastDateUpdated)));
        statement.setString(6, (this.lastUpdatedByUser == null ? userId
: this.lastUpdatedByUser));
        statement.setString(7, this.problemId);
        statement.setString(8, this.udfId);
        statement.executeUpdate();
    }
}

} catch (Exception e) {
    if (outErrMsg != null) {
        throw e;
    }
    ErrorWriter.write(e, ErrorWriter.LOGERR);
} finally {
    try {
        if (statement != null) statement.close();
    } catch (Exception e) {} // pass
}
}

/**
 * getOneUdfValue gets the value from a problem_udf record(in HashMap)
 * given the udfId and displayType
 */
public static String getOneUdfValue(Map rowData, String udfId, String
displayType) throws Exception {
    String newValue = null;

    try {
        if ("LIST".equals(displayType) || "POPUP".equals(displayType) ||
"TAB".equals(displayType)){
            newValue = (String) rowData.get("UDF_LIST_ID");

        } else if (displayType.equals("NUMBER")) {
            newValue = (String) rowData.get("VALUE_NUMBER");

        } else if (displayType.equals("DATE")) {
            newValue = (String) rowData.get("VALUE_DATE");

        } else {
            newValue = (String) rowData.get("VALUE");

        }
    } catch (Exception e) {
        ErrorWriter.write(e, ErrorWriter.LOGERR);
        throw e;
    }
    return newValue;
} //end getOneUdfValue

/**
 * checks if the udf passed in is the same
 */
public boolean issame(Problem_UDF pu) throws Exception{
    if (this.udfId != pu.udfId) return false;
    if (this.problemId != pu.getProblemId()) return false;

    DDEntry dde = Z.dictionary.getDDEntry(dbconn, name);

```

Problem_UDF.java

```

        if (Z.dictionary.isListType(dde) && this.udfListId != pu.udfListId) return
false;
        else if (Z.dictionary.isProblemTextType(dde)){
            if (this.getIdSeq() != pu.getIdSeq() || this.getTextSeq() !=
pu.getTextSeq()) return false;
            if (this.text != pu.text) return false;
        }
        else if ("DATE".equals(dde.getType()) && this.valueDate != pu.valueDate)
return false;
        else if (this.value != pu.value) return false;

        return true;
    }

    /**
     * setUdfHash set all udf in a hash in this form (ddname, value).
     */
    public static void setUdfHash(Connection dbconn, SesameSession session, String
problemId, HashMap udfHm) throws Exception {
        String value = "";
        Problem_UDF udf = null;
        String uName = null;
        String displayType = null;
        ArrayList udfList = new ArrayList();
        StringBuffer v = new StringBuffer();
        String sep = ";";

        try {
            long t = System.currentTimeMillis();
            udfList = Problem_UDF.getReference(dbconn, problemId);
            //loop thru thisUdfs to get each row of udf info.
            for(int i=0; i<udfList.size(); i++) {
                udf = (Problem_UDF)udfList.get(i);
                displayType = udf.getUDFDisplayType();
                //get the value by displayType
                Z.log.writeToLog(Log.DEBUG, "setUdfHash name: " + udf.getName() + "
displayType:" + displayType);
                if ("LIST".equals(displayType)) {
                    if (udfHm.get(udf.getName()) != null) { //check if we have gone thru
this udf already.
                        //check if it's multivalues
                        DDEntry dde = Z.dictionary.getDDEntry(dbconn, udf.getName());
                        if (dde.getMultipleValue().equals("Y")) {
                            ArrayList vList = Problem_UDF.getUdfListValues(dbconn,
problemId, udf.getUdfId());
                            for (int j=0; j<vList.size(); j++) {
                                v.append(vList.get(j));
                                v.append(sep);
                            }
                            value = removeLast(v.toString());
                            v.setLength(0);
                            //Z.log.writeToLog(Log.INFO, "multiple udf value " + value);
                        }
                    }
                }
                else{
                    value = udf.getUdfListValue(dbconn, problemId, udf.getUdfId());
                }
            }
            else if ( "TEXTFIELD".equals(displayType) ||
"URL".equals(displayType) || "BOOLEAN".equals(displayType) ||
"POPUP".equals(displayType) ) {
                value = udf.getValue();
            }
        }
    }

```

```

                                Problem_UDF.java
        }else if ( "NUMBER".equals(displayType) ) {
            value = udf.getValueNumber();
        }else if ( "DATE".equals(displayType) ) {
            calendar cal = udf.getValueDate();
            // value = Display.formatDate(session, dbconn,
udf.getValueDate());
            // let's get the date in the right mask for mdr
            SimpleDateFormat sdf = new SimpleDateFormat("MM/dd/yy");
            value = sdf.format(cal.getTime());
        }else if ( "TEXTAREA".equals(displayType) ||
"PRINTTEXT".equals(displayType) ) {
            value = udf.getText();
        }else if ( "LOGAREA".equals(displayType) ) {
            value = udf.getText();
        }//end type
        //put value in hashmap
        udfHm.put(udf.getName(), value);
    }//end for

    getAllUdfs(udfHm);
    long t2 = System.currentTimeMillis();
    Z.probe("***Time to get udf Hash***" + (t2 - t) );

```

```

    }catch (Exception e) {
        ErrorWriter.write(e, ErrorWriter.LOGERR);
        throw e;
    }//end try
}//end setUdfHash

```

```

private static String removeLast(String s) {
    //remove the last ;
    int len = s.length();
    String str = "";
    if (len > 1) str= s.substring(0, len - 1);
    return str;
}//end removeLast

```

```

/**
 * getAllUdf gets the udf names from dd to fill out
 * those that have no values in the form.
 */

```

```

private static void getAllUdfs(HashMap udfHm) throws Exception {

```

```

    HashMap dict = DataDictionary.getHashMap();
    Iterator iter = dict.keySet().iterator();
    while (iter.hasNext()) {
        String key = (String)iter.next();
        DDEntry dde = (DDEntry)dict.get(key);
        if (("UDF".equals(dde.getType()) && (udfHm.get(key) == null)))
            udfHm.put(key, "");
    }

```

```

}//end getAllUdfs

```

```

public String toDisplayString(SesameSession session) throws Exception {
    SecurityUser su = (SecurityUser)session.getAttribute("USER");
    Locale locale = session.getLocale();
    Z.log.writeToLog(Z.log.DEBUG, "PU: name/displaytype is: " + this.getName() + "/"
+ udfDisplayType + "/" + this.getUDFDisplayType());
    if ("DATE".equals(udfDisplayType)) {
        DbTime dbt = new DbTime(session, getValueDate());
        return dbt.getFormattedDate(su.getDateFormat());
    }

```

Problem_UDF.java

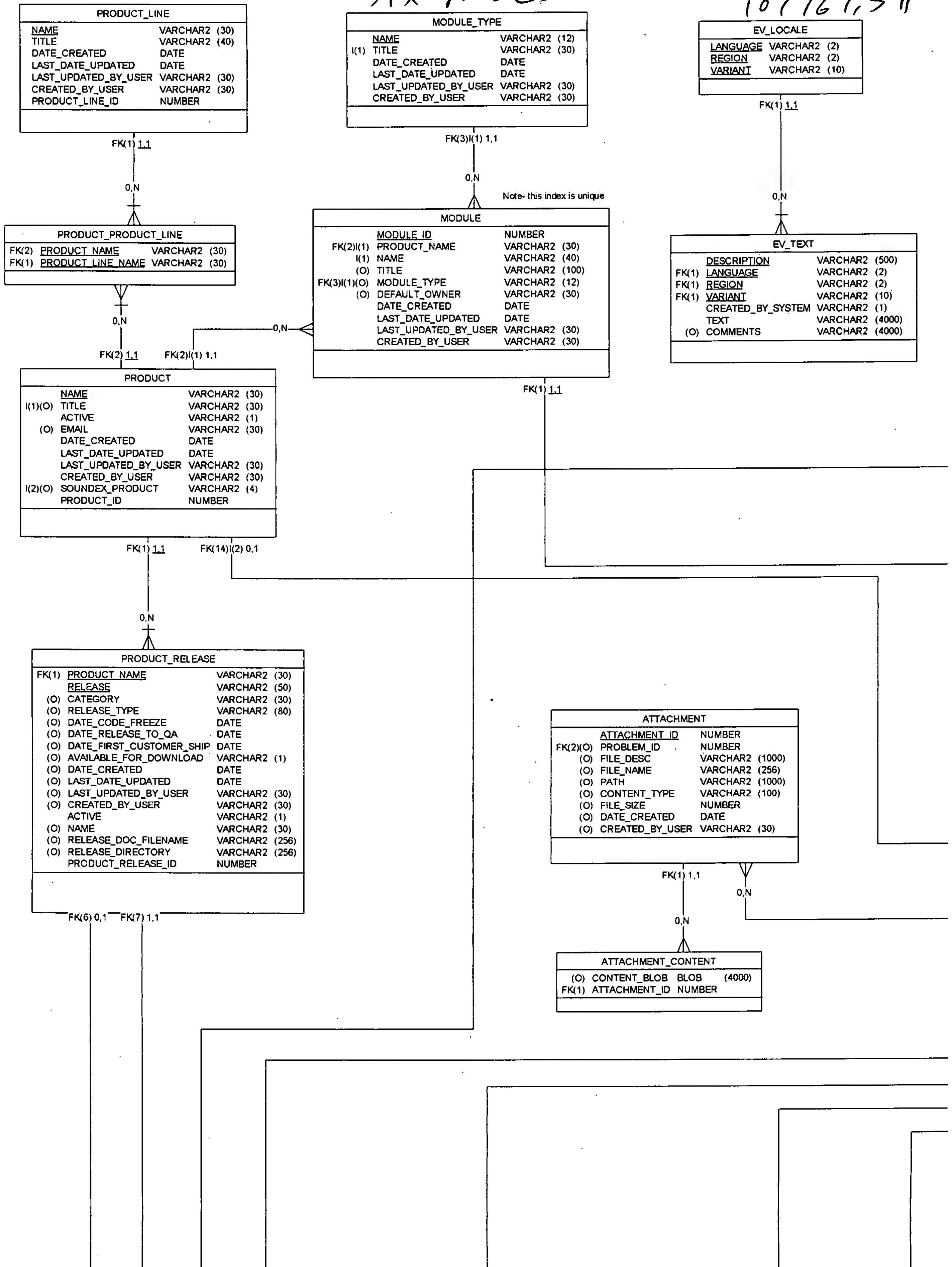
```

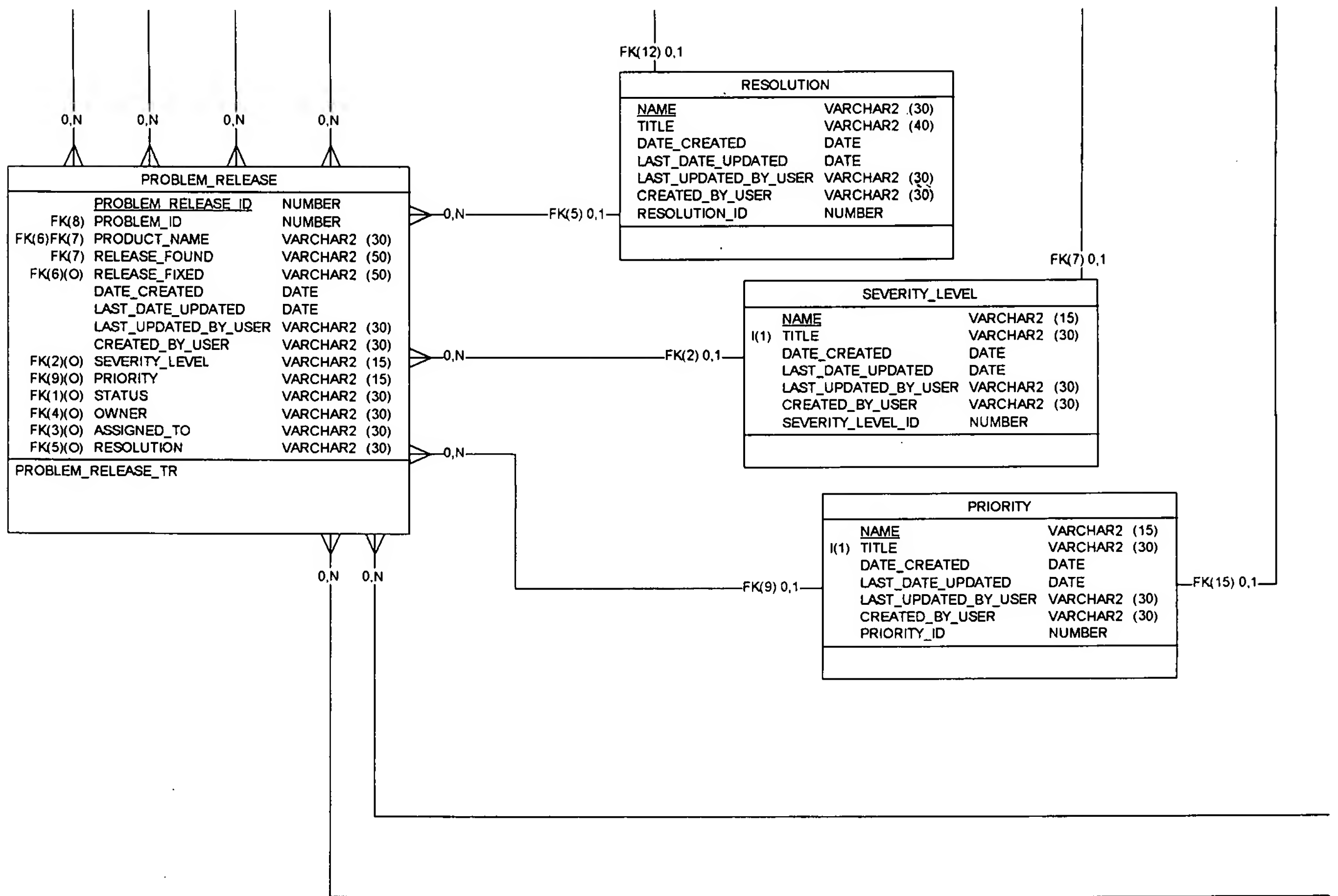
}
else if ("IMAGE".equals(udfDisplayType))
    return null;
else if ("LABEL".equals(udfDisplayType))
    return getText();
else if ("LIST".equals(udfDisplayType)) {
    UdfList ul = UdfList.getReference(dbconn, udfListId);
    Z.log.writeToLog(Z.log.DEBUG, "PU: udf list value is:" + ul.getTitle());
    return ul.getTitle();
}
else if ("LOGAREA".equals(udfDisplayType))
    return getText();
else if ("NUMBER".equals(udfDisplayType)) {
    NumberFormat form = NumberFormat.getInstance(locale);
    return form.format(valueAsNumber);
}
else if ("POPUP".equals(udfDisplayType))
    return getUdfListValue(dbconn, problemId, udfId);
else if ("PRINTTEXT".equals(udfDisplayType))
    return getText();
else if ("TAB".equals(udfDisplayType))
    return getUdfListValue(dbconn, problemId, udfId);
else if ("TEXTAREA".equals(udfDisplayType))
    return getText();
else if ("TEXTFIELD".equals(udfDisplayType))
    return getText();
else if ("USER".equals(udfDisplayType))
    return SecurityUser.getUserNameDisplay( dbconn, value, false );
else if ("RADIOBUTTON".equals(udfDisplayType))
    return null; // ??
else if ("CURRENCY".equals(udfDisplayType)) {
    NumberFormat form = NumberFormat.getCurrencyInstance(locale);
    return form.format(valueAsNumber);
}
else if ("DECIMAL".equals(udfDisplayType)) {
    NumberFormat form = NumberFormat.getInstance(locale);
    return form.format(valueAsNumber);
}
return null;
}
}

```

EXHIBIT F SIX PAGES

REF: US PAT APPN.
SER NO.
101767,511





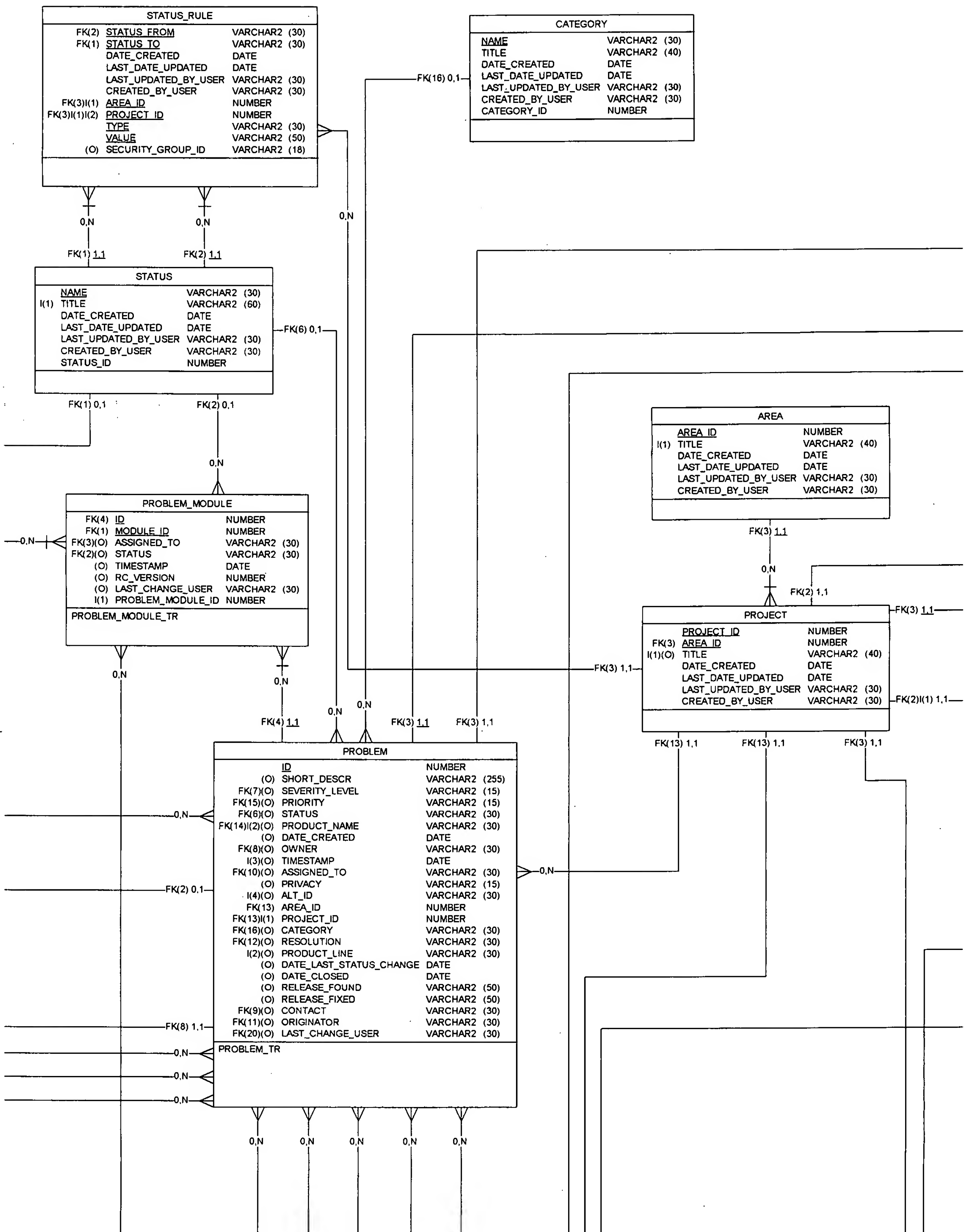
PROBLEM_HIST		
I(1)I(2)	ID	NUMBER
(O)	SHORT_DESCR	VARCHAR2 (255)
(O)	SEVERITY_LEVEL	VARCHAR2 (15)
(O)	PRIORITY	VARCHAR2 (15)
I(2)(O)	STATUS	VARCHAR2 (30)
(O)	PRODUCT_NAME	VARCHAR2 (30)
(O)	DATE_CREATED	DATE
(O)	OWNER	VARCHAR2 (30)
(O)	TIMESTAMP	DATE
(O)	ASSIGNED_TO	VARCHAR2 (30)
(O)	PRIVACY	VARCHAR2 (15)
(O)	LAST_CHANGE_USER	VARCHAR2 (30)
(O)	ALT_ID	VARCHAR2 (30)
(O)	AREA_ID	NUMBER
(O)	PROJECT_ID	NUMBER
(O)	CATEGORY	VARCHAR2 (30)
(O)	RESOLUTION	VARCHAR2 (30)
(O)	PRODUCT_LINE	VARCHAR2 (30)
I(3)(O)	DATE_LAST_STATUS_CHANGE	DATE
(O)	DATE_CLOSED	DATE
(O)	RELEASE_FOUND	VARCHAR2 (50)
(O)	RELEASE_FIXED	VARCHAR2 (50)
(O)	CONTACT	VARCHAR2 (30)
(O)	CHANGE_TYPE	VARCHAR2 (1)
(O)	HIST_TIMESTAMP	DATE
(O)	ORIGINATOR	VARCHAR2 (30)

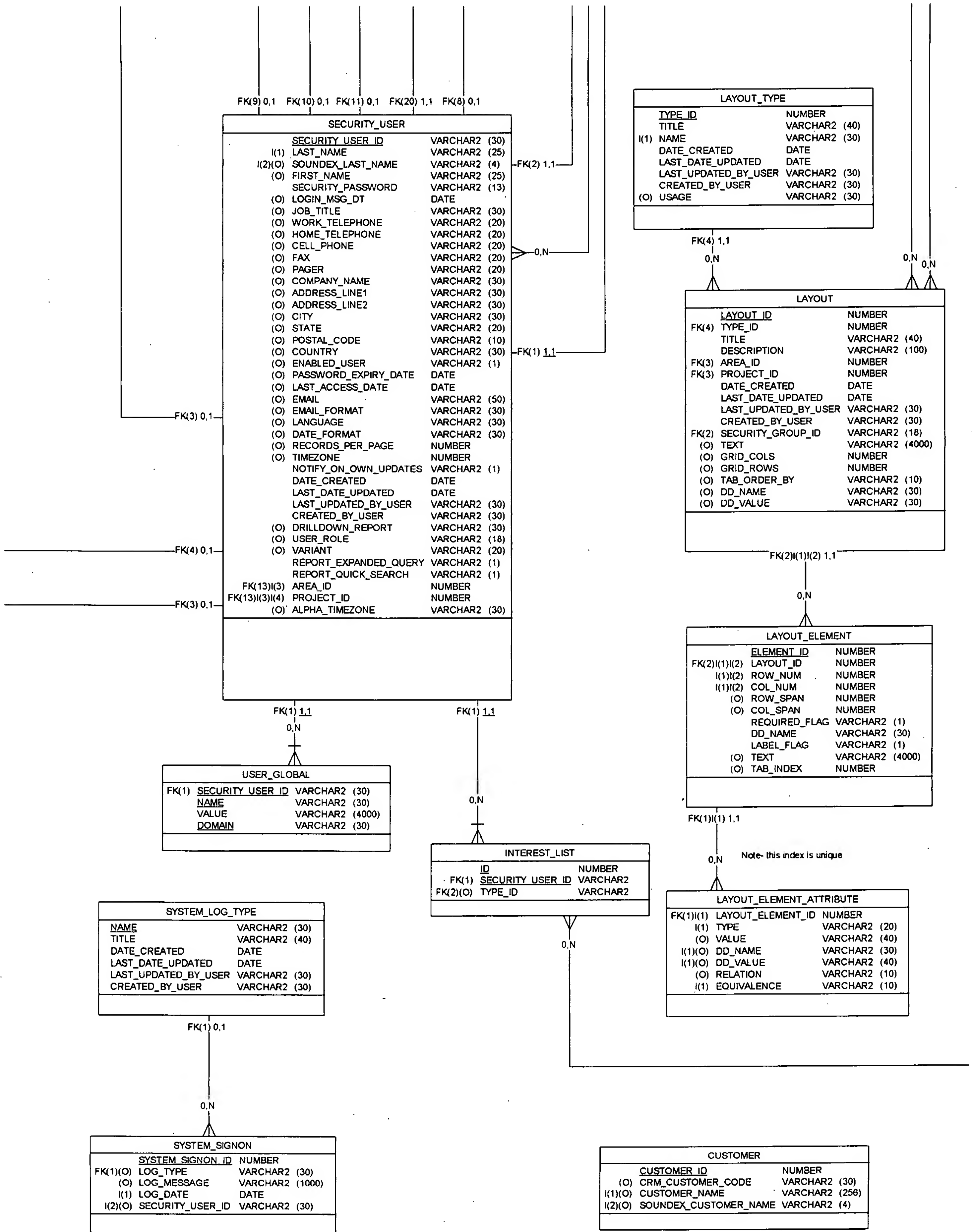
PROBLEM_MODULE_HIST		
I(1)	ID	NUMBER
(O)	MODULE_ID	NUMBER
(O)	ASSIGNED_TO	VARCHAR2 (30)
(O)	STATUS	VARCHAR2 (15)
(O)	TIMESTAMP	DATE
(O)	RC_VERSION	NUMBER
(O)	LAST_CHANGE_USER	VARCHAR2 (30)
(O)	CHANGE_TYPE	VARCHAR2 (1)
(O)	HIST_TIMESTAMP	DATE
(O)	PROBLEM_MODULE_ID	NUMBER

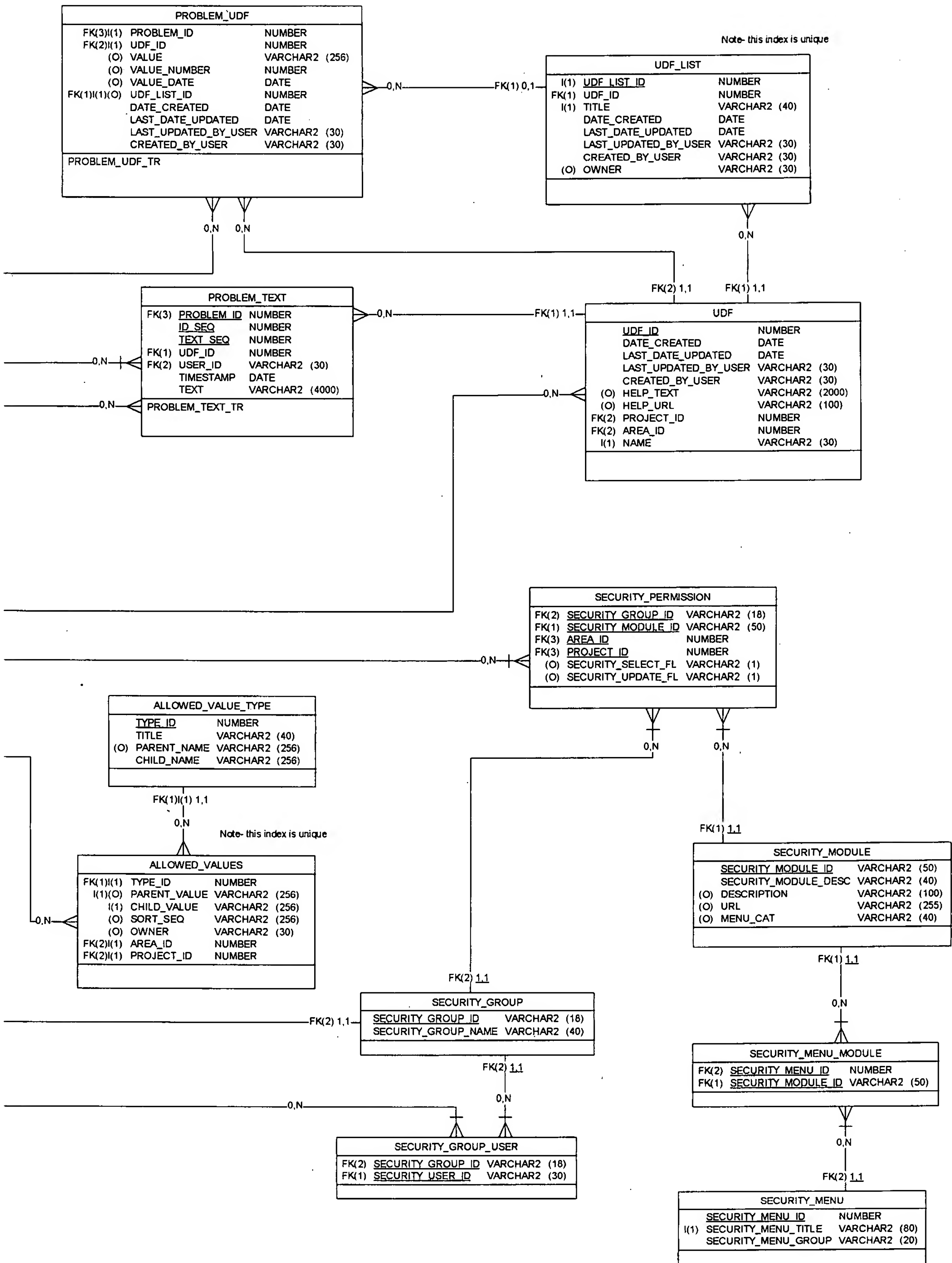
PROBLEM_TEXT_HIST		
I(1)I(2)	PROBLEM_ID	NUMBER
(O)	UDF_ID	NUMBER
I(2)	ID_SEQ	NUMBER
I(2)	TEXT_SEQ	NUMBER
(O)	USER_ID	VARCHAR2 (30)
(O)	TIMESTAMP	DATE
(O)	TEXT	VARCHAR2 (4000)
(O)	CHANGE_TYPE	VARCHAR2 (1)
(O)	HIST_TIMESTAMP	DATE

PROBLEM_RELEASE_HIST		
I(1)	PROBLEM_ID	NUMBER
(O)	PRODUCT_NAME	VARCHAR2 (30)
(O)	RELEASE_FOUND	VARCHAR2 (50)
(O)	RELEASE_FIXED	VARCHAR2 (50)
(O)	DATE_CREATED	DATE
(O)	LAST_DATE_UPDATED	DATE
(O)	LAST_UPDATED_BY_USER	VARCHAR2 (30)
(O)	CREATED_BY_USER	VARCHAR2 (30)
(O)	SEVERITY_LEVEL	VARCHAR2 (15)
(O)	PRIORITY	VARCHAR2 (15)
(O)	STATUS	VARCHAR2 (30)
(O)	OWNER	VARCHAR2 (30)
(O)	ASSIGNED_TO	VARCHAR2 (30)
(O)	RESOLUTION	VARCHAR2 (30)
(O)	CHANGE_TYPE	VARCHAR2 (1)
(O)	HIST_TIMESTAMP	DATE
(O)	PROBLEM_RELEASE_ID	NUMBER

PROBLEM_UDF_HIST		
I(1)	PROBLEM_ID	NUMBER
(O)	UDF_ID	NUMBER
(O)	VALUE	VARCHAR2 (256)
(O)	VALUE_NUMBER	NUMBER
(O)	VALUE_DATE	DATE
(O)	UDF_LIST_ID	NUMBER
(O)	DATE_CREATED	DATE
(O)	LAST_DATE_UPDATED	DATE
(O)	LAST_UPDATED_BY_USER	VARCHAR2 (30)
(O)	CREATED_BY_USER	VARCHAR2 (30)
(O)	CHANGE_TYPE	VARCHAR2 (1)
(O)	HIST_TIMESTAMP	DATE







DATA_DICTIONARY	
NAME	VARCHAR2 (30)
TITLE	VARCHAR2 (40)
(O) HELP_TEXT	VARCHAR2 (2000)
(O) HELP_URL	VARCHAR2 (100)
REQUIRED_FIELD	VARCHAR2 (1)
DATE_CREATED	DATE
LAST_DATE_UPDATED	DATE
LAST_UPDATED_BY_USER	VARCHAR2 (30)
CREATED_BY_USER	VARCHAR2 (30)
TYPE	VARCHAR2 (30)
(O) DEFAULT_SIZE	NUMBER
(O) DEFAULT_FORMAT	VARCHAR2 (40)
(O) DEFAULT_VALUE	VARCHAR2 (100)
(O) DISPLAY_FORMAT	VARCHAR2 (10)
(O) DISPLAY_TYPE	VARCHAR2 (10)
(O) PRIMARY_SQL	VARCHAR2 (4000)
(O) DEPENDENT_SQL	VARCHAR2 (4000)
(O) ORDER_BY_SQL	VARCHAR2 (500)
(O) PARENT1_FIELD_NAME	VARCHAR2 (30)
(O) PARENT1_SQL	VARCHAR2 (4000)
(O) PARENT2_FIELD_NAME	VARCHAR2 (30)
(O) PARENT2_SQL	VARCHAR2 (4000)
DISPLAY_AS_URL	VARCHAR2 (1)
(O) URL	VARCHAR2 (1000)
SELECT_FOR_REPORTS	VARCHAR2 (1)
(O) TABLE_NAME	VARCHAR2 (30)
(O) COLUMN_NAME	VARCHAR2 (30)
(O) ALLOW_ADD_TO_LIST	VARCHAR2 (1)
(O) LOOKUP_TABLE	VARCHAR2 (30)
(O) LOOKUP_KEY	VARCHAR2 (30)
(O) LOOKUP_COLUMN1	VARCHAR2 (200)
(O) LOOKUP_COLUMN2	VARCHAR2 (30)
(O) LOOKUP_COLUMN3	VARCHAR2 (30)
FILTER_CRITERIA	VARCHAR2 (1)
(O) PARENT_TABLE	VARCHAR2 (30)
(O) PARENT_KEY	VARCHAR2 (30)
(O) CHILD_KEY	VARCHAR2 (30)
(O) ENABLE_INTEREST_LIST	VARCHAR2 (1)
IS_SORTABLE	VARCHAR2 (1)
REMEMBER_LAST_VAL	VARCHAR2 (1)
MULTIPLE_VALUE	VARCHAR2 (1)
(O) LOOKUP_ID	VARCHAR2 (30)

SORT_ORDER	
<u>SORT_ORDER_ID</u>	NUMBER
TITLE	VARCHAR2 (40)
DESCRIPTION	VARCHAR2 (100)

OUTPUT_TYPE	
TITLE	VARCHAR2 (40)
DESCRIPTION	VARCHAR2 (100)
DATE_CREATED	DATE
LAST_DATE_UPDATED	DATE
LAST_UPDATED_BY_USER	VARCHAR2 (30)
CREATED_BY_USER	VARCHAR2 (30)
NAME	VARCHAR2 (15)

SORT_ORDER_FIELD	
<u>SORT_ORDER_FIELD_ID</u>	NUMBER
FK(2)(1) SORT_ORDER_ID	NUMBER
FK(1)(1) DD_NAME	VARCHAR2 (30)
(O) ORDER_RANK	NUMBER
(O) SORT_DIRECTION	VARCHAR2 (4)

REPORT	
<u>REPORT_ID</u>	NUMBER
LAYOUT_ID	NUMBER
(O) SORT_ORDER_ID	NUMBER
FK(3) FILTER_GROUP_ID	NUMBER
(O) REPORT_GROUP_ID	NUMBER
TITLE	VARCHAR2 (40)
DESCRIPTION	VARCHAR2 (100)
SECURITY_GROUP_ID	VARCHAR2 (18)
SECURITY_USER_ID	VARCHAR2 (30)
DATE_CREATED	DATE
LAST_DATE_UPDATED	DATE
LAST_UPDATED_BY_USER	VARCHAR2 (30)
CREATED_BY_USER	VARCHAR2 (30)
FK(2)(O) OUTPUT_TYPE_NAME	VARCHAR2 (15)
REPORT_TYPE_NAME	VARCHAR2 (15)

FILTER_GROUP	
<u>FILTER_GROUP_ID</u>	NUMBER
TITLE	VARCHAR2 (40)
DESCRIPTION	VARCHAR2 (100)
(O) FILTER_GROUP_TYPE	VARCHAR2 (1)

FILTER	
<u>FILTER_ID</u>	NUMBER
FK(3)(1) FILTER_GROUP_ID	NUMBER
FK(2)(1) DD_NAME	VARCHAR2 (30)

FILTER_CRITERIA	
<u>FILTER_CRITERIA_ID</u>	NUMBER
FK(1) FILTER_ID	NUMBER
FILTER_CRITERIA_VALUE	VARCHAR2 (40)

USER_SESSION	
<u>SESSION_ID</u>	VARCHAR2 (30)
SECURITY_USER_ID	VARCHAR2 (30)
EXPIRE_DATE	DATE
(O) USER_ROLE	VARCHAR2 (18)
(O) AREA_ID	NUMBER
(O) PROJECT_ID	NUMBER
(O) SITE_URL	VARCHAR2 (100)

EV_SEQUENCE	
NAME	VARCHAR2 (40)
(O) MIN_VALUE	NUMBER
(O) MAX_VALUE	NUMBER
(O) INCREMENT_BY	NUMBER
(O) CACHE_SIZE	NUMBER
(O) LAST_VALUE	NUMBER

DUAL	
I(1)(O) DUM	VARCHAR2 (1)

EV_CACHE	
<u>EV_CACHE_ID</u>	VARCHAR2 (40)
(O) CACHE_DATA	BLOB (4000)
(O) TIMESTAMP	DATE

APPLICATION_DEFAULT	
NAME	VARCHAR2 (30)
(O) VALUE	VARCHAR2 (4000)
(O) COMMENTS	VARCHAR2 (64)

Schema Name	
ExtraView Release 4.0 build 1.112	
Creator	Version
Sesame Technology	1.0
Created	Modified
04/01/2001	03/11/2002

ExtraView Objects Definition

April 27, 2001

Overview

This document defines the current set of “significant” objects that will be created for the ExtraView Java (EVJAVA) product.

Object List

In the following bulleted list, we list the object by name. If there are any pertinent inheritance relationships, then the phrase “:is a <parent-object-name>” is listed next to the object’s name. The phrase “:has:” or “& has:” indicates that the attributes or relationships in the underlying bulleted list belong to the object.

When a relationship is described, the cardinality is expressed as (1:x) or (1:x..y) where x is the minimum number of related objects and y is the maximum number of related objects.

These objects do not map 1-1 to the tables in ExtraView. However, the objects were chosen so as to imply an obvious mapping to the underlying data. There may be some schema changes required to support these objects robustly.

This is a partial list, and it will be augmented over time; so keep your eye on the date in the title. In particular, right now, the security objects and administration objects are not well-defined.

A “(?)” after an attribute or relationship indicates that the field is currently not being used, or it is being redesigned, or its meaning is undecipherable from the PL/SQL code and the existing schema.

Note: To reduce clutter, I have factored out the following attributes: Date Created, Date Last Updated, Created By User, and Updated By User. These attributes should be part of any table that the user can update, and they will be factored in later (somehow).

Application Logic Objects

Application logic objects provide the methods necessary for the application logic behavior, in addition to the attributes that are maintained in the database for the object. The application logic objects should not receive or send HTML, nor should it directly access the database. The Presentation Tier and the DBMS Tier protect the application logic objects from these external influences.

Most application logic objects are persistable. The persistence is explicitly invoked for each object; every persistable object implements the Persist interface. This is similar to Java’s Externalizable interface implementation, but the Persist interface gives the application full control over which objects are persisted, whereas the Externalizable implementation tends to remove this control from the application.

- Problem: has:



-
- Synopsis
 - Priority (1:1 mapping to Priority)
 - Severity (1:1 mapping to Severity Level)
 - Resolution (1:1 mapping to Resolution)
 - Originator (= 1:1 relationship to User)
 - Component Container (= 1:0..1 relationship to Component)¹
 - Platform Affected (= 1:0..1 relationship to Platform)²
 - Product Affected (= 1:0..1 relationship to Product)³
 - Releases Affected (= 1:0..n relationship to Release)
 - Description
 - Comments
 - Workaround
 - Release notes
 - (1:0..n) Enclosure(s) (maintained in Enclosure table)
 - (1:0..1) Product Release found
 - (1:0..1) Product Release fixed
 - UDF Values (1:0..n relationship to UDF Value)
 - User
 - Security User Id
 - Last Name
 - Soundex Last Name
 - First Name
 - Security Password
 - Login Message Date
 - Job Title
 - Work Telephone
 - Home Telephone
 - Cell Phone
 - Fax
 - Pager
 - Company Name
 - Address Line1
 - Address Line2
 - City
 - State
 - Postal Code
 - Country
 - Flags (Enabled/Disabled, Notify-On-Own-Updates/Do-Not-Notify)
 - Password Expiry Date
 - Last Access Date
 - Email

¹ 1:0..n in future

² 1:0..n in future

³ 1:0..n in future



-
- Email Format
 - Locale
 - Date Format (pattern as used by DateFormatter)
 - Records Per Page
 - Time Zone
 - Group Participation (= 1:0..n relationship to Security Group)
 - Problem Interest Lists (=1:0..n relationship to Problem Interest List)
 - Module Interest Lists (=1:0..n relationship to Module Interest List)
 - Product Interest Lists (=1:0..n relationship to Product Interest List)
 - Interest List: an abstract base class
 - Users (=1:0..n relationship to User)
 - Problem Interest List is an Interest List & has:
 - Problem (1:1 relationship to Problem)
 - Module Interest List is an Interest List & has:
 - Module (1:1 relationship to Module)
 - Product Interest List is an Interest List & has:
 - Product (1:1 relationship to Product)
 - Titled Object: an abstract base class
 - ID
 - Name
 - Title
 - Sort sequence (within TitledObjects with same ID)
 - User-Defined Field (aka UDF): is a Titled Object & has:
 - Data Type (Text Field, Text Area, Log Area, Number, Date, List, or URL)
 - Flags:
 - required/optional,
 - display-as-url/display-normal,
 - select-on-reports/no-select-on-reports
 - Help Text
 - Help URL
 - UDF Values (1:0..n relationship to UDF Value)
 - UDF Value has:
 - Problem Id
 - UDF Id
 - Value
 - Value Number
 - Value Date
 - UDF List Membership (1:1 relationship to UDF List)
 - UDF List : has
 - Title
 - Owning User
 - (1:0..n relationship to UDF)
 - Component: is a UDF
 - Platform: is a UDF



- Product: is a UDF
- Description: is a UDF
- Comments: is a UDF
- Workaround: is a UDF
- Release Notes: is a UDF
- Product Line: is a Titled Object & has:
 - Products in the Line (1:0..n relationship to Product)
- Problem Release (defines 1:0..n relationship of Problem to Product Release)
 - Problem ID
 - Product
 - Release Found
 - Release Fixed
 - Short Description
 - Severity Level
 - Priority
 - Status
 - Owner
 - Timestamp
 - Assigned To
 - Hours Billed
 - Privacy
 - Full Description
 - Eng Remarks
 - QA Remarks
 - Alternate Id
 - Public Description
 - Area Id
 - Project Id
 - Category
 - Resolution
 - Product Line Membership (= 1:0..1 relationship to Product Line)
 - Date Last Status Change
 - Date Closed
 - Contact
 - Originator
- Report
 - Report Name
 - Report Owner
 - Included Report Columns (1:0..n relationship to Report Column)
 - Sort sequence (1:0..4 relationship to Report Column)
- Report Column
 - Data Dictionary Element
 - Sort Sequence Number
 - Heading



- Flags (break/no-break)
 - Format (?)
- Message
 - Description (always in English)
 - Locale Language
 - Locale Region
 - Locale Variant
 - Text (Unicode): as a MessageFormat pattern
 - Comment
 - Flags (translate/no translate)
- Icon
 - Description (always in English)
 - Locale Language
 - Locale Region
 - Locale Variant
 - Graphic content
 - Comment
 - Flags (translate/no translate)
- Enclosure:
 - ID
 - Title
 - Type (user-defined enumeration value)
 - Text (LOB)
- Product: is a Titled Object & has:
 - Flags (active/inactive)
 - Email Address
 - Soundex
- Product Release: is a Titled Object & has
 - Defining Product (=1:1 relationship to Product)
 - Category Membership (=1:0..1 relationship to Category)
 - Release Type (?)
 - Date Of Code Freeze
 - Date Released To QA
 - Date First Customer Ship
 - Flags (Active/Inactive, Available-For-Download/Unavailable-For-Download)
 - Release Document Filename (?)
 - Release Directory
- Module: is a Titled Object & has:
 - Defining Product (= 1:1 relationship to Product)
 - Type (= 1:1 relationship to Module Type)
 - Default Owner (= 1:0..n relationship to User)
- Module Type: is a Titled Object
- Category: is a Titled Object



-
- Resolution: is a Titled Object
 - Severity Level: is a Titled Object
 - Status: is a Titled Object
 - Project: is a Titled Object & has:
 - Area membership (= 1:0..n relationship to Area)
 - Area: is a Titled Object
 - Data Dictionary Element
 - Name
 - Title
 - Help Text
 - Help Url
 - Flags :
 - Required/optional,
 - Display-as-url/display-normal,
 - select-for-reports/no-select-for-reports
 - Type
 - Default Size
 - Default Format
 - Default Value
 - Display Format
 - Display Type
 - Primary SQL
 - Dependent SQL
 - Order By SQL
 - Parent1 Field Name (?)
 - Parent1 SQL (?)
 - Parent2 Field Name (?)
 - Parent2 SQL (?)
 - Url
 - Referenced Table Name (?)
 - Column Name (?)
 - Security Group: is a Titled Object & has:
 - User Members (1:0..n relationship to User)
 - Select Resource Permissions (1:0..n relationship to Security Resource) = "is permitted to select"
 - Update Resource Permissions (1:0..n relationship to Security Resource) = "is permitted to update"
 - Security Resource (aka "Security Module")
 - ID
 - Short Description
 - Long Description
 - Application Defaults
 - Property Settings (1:1 relationship to Property Set)



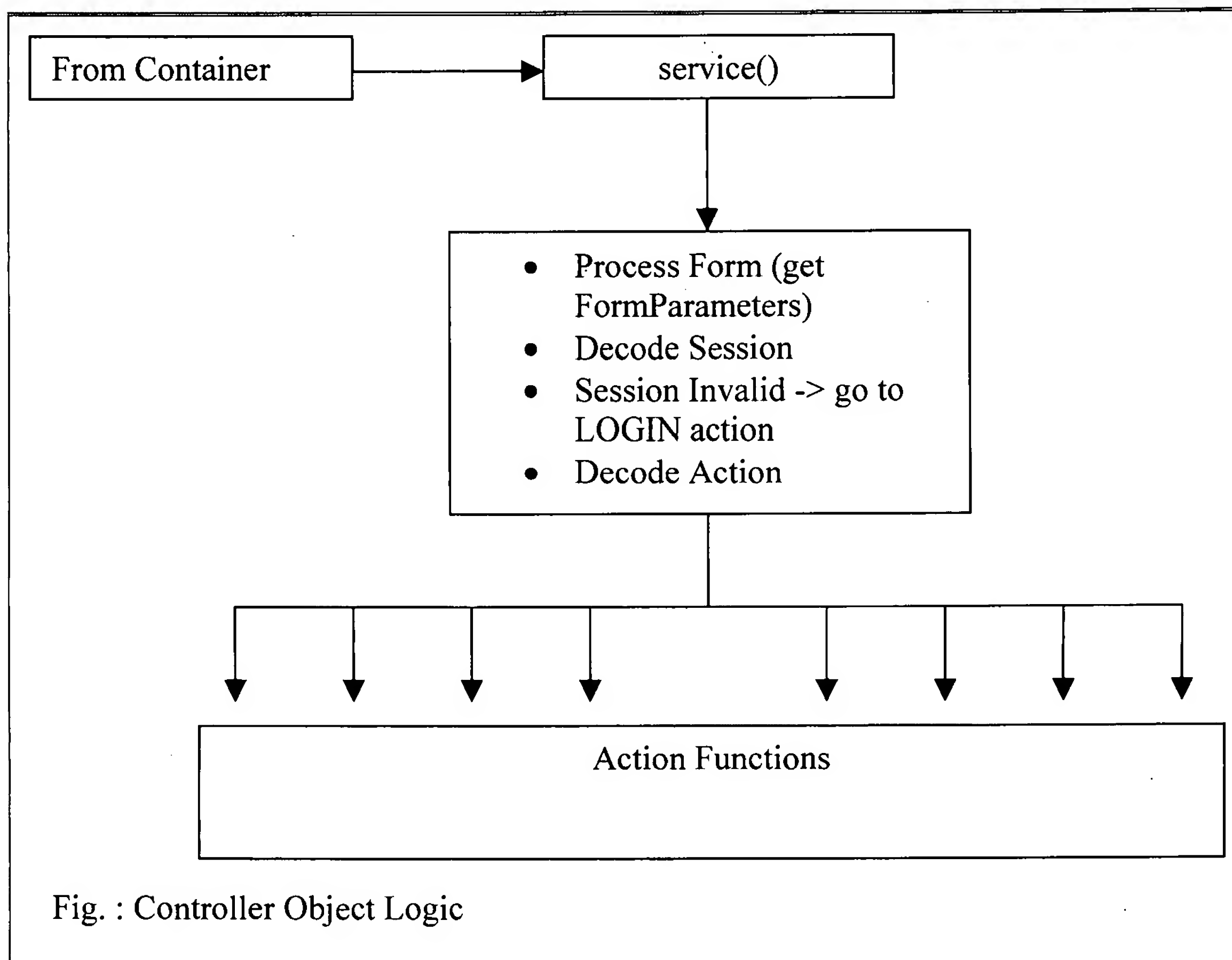
-
- Allowed Values (this needs work... imposes hierarchy and constraints on object attribute values)
 - Type (1:1 mapping to Allowed Value Type)
 - Parent Allowed Value
 - Child Allowed Value
 - User Globals
 - Security User
 - Current Name Value Pairs (1:1 relationship to Property Set)
 - Template (Sesame infrastructure object)
 - Session
 - JSessionId
 - Property Sets (1:0..n relationship to Property Set)
 - Property Set
 - Name-Value Pairs (1:0..n relationship to Name Value Pair)

Presentation Objects

The Presentation Servlet Tier contains the objects that are responsible for rendering and collecting data from forms sent to/from the browser. This tier has its own set of objects which have behavior and lifetimes different from the objects in the application Logic Tier. Many of the presentation objects have counterparts in the application logic tier. For example, the ProblemDisplay object is directly related to the Problem object. However, a Problem object has a long lifetime in the database, whereas the ProblemDisplay object lives only for the period of time it takes to interact with the user to display, update, or insert the object.

The Controller object is the top-level action routine that is called to decode and execute whatever action is required by the submitted form parameters. In one sense, it is a big “switch” that maps the form to an internal action. The following diagram illustrates a suggested implementation of the Controller.





FormParameters is a transient (non-persistent) object that is re-initialized anew for each new form that is submitted. It is a wrapper for the string of parameter values that are submitted with the form. Action routines are responsible for the decoding of the parameter values (except for those used by the Controller) in FormParameters. Most of the presentation objects are transient and can be discarded for each round-trip with the client. However, some object state needs to be kept on a per-session basis. The SessionContext object can be used to persist state values for the lifetime of the session. The SessionContext persisted objects in permanent storage must be garbage-collected on an occasional basis, since sessions may disappear quietly.

The top-level objects are:

- Controller
- My Home Display
- Add Problem Display
- Search Report Display
- Administration Display
- Help Display
- Sign Off Display
- Session Context
- Form Parameters



October 16, 2007

Administration and Configuration Display Objects:

- 4 User and Security Objects
- 5 Batch Command Objects
- 4 Business Rule Objects
- 10 Installation and Setup Objects
- 10 Configuration Objects

Add Problem Display Objects:

- New Problem Summary Display
- Interest List Display
- Edit Problem Display

Search Report Display Objects:

- Quick List Display
- Full Detail Report
- Summary Report
- Expanded Query Display
- Save Query Popup
- Additional Reports Display
- Edit Personal Report
- Add Personal Report
- Load Saved Query

Help Objects:

- ExtraView Help Index Popup
- Static Help Display

DBMS Objects

The DBMS objects implement the DBMS java interfaces that are necessary to support the persistence and query functions of the other tiers. The implementing classes are determined and loaded at runtime, when the database type and instance are defined in the standard Sesame properties file.

Most of the database independence arises from the use of JDBC for database access.

However, there are database dependencies in many of the operations performed, e.g., in the SQL statements. These can be avoided through the use of ANSI-99 entry-level SQL in most cases. The DBMS objects are defined so as to provide support for any required functions that are not database-independent.

The top-level object is the DBMS object, which is a wrapper for all of the methods and attributes accessible in the DBMS. The standard Sesame infrastructure to obtain connections and manage the database resources is already database-independent; therefore, the Sesame framework is used for connection management. The DBMS object is used for database functions that differ from one database to another.

- DBMS: Controls access to the database; loads the classes responsible for implementing the DBMS interfaces. Representative methods are:
 - InitializeDBMS: sets up initial parameters and classes based on Sesame properties configuration entries.



-
- CreateSequence: creates a named database object that maintains a counter for use in defining unique column values (like the Oracle Sequence construct).
 - GetSequenceNextValue: bumps the sequence counter by the sequence increment and returns the sequence counter value.
 - LOB functions (?)
 - Others tbd



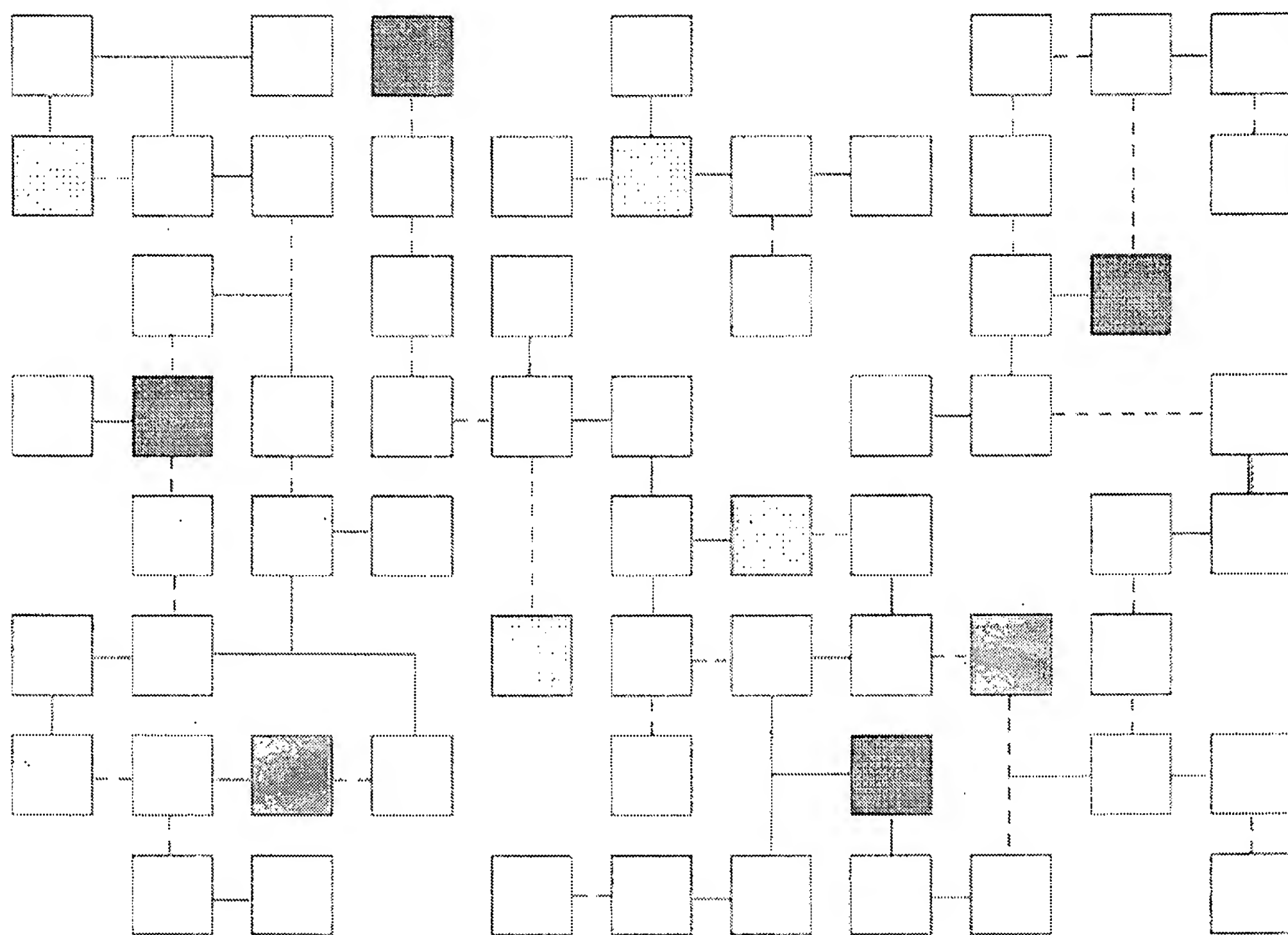
October 16, 2007

REF: U.S. PATENT APPLICATION
SER NO. 10/ 767,511

EXHIBIT B

89 PAGES.

ExtraView™ Schema Version 3.1.2.1





Sesame Technology
269 Mount Hermon Road, Suite 205
Scotts Valley, CA 95066

Telephone: (831) 461-7100
Fax: (831) 461-7104
E-mail: info@sesame.com
www.sesame.com
© 1999 - 2001 Sesame Technology
All rights reserved

Manual Name: ExtraView Schema
Revision Date: September 15, 2001

Information contained in this document, and the software to which it refers is subject to change without notice. This includes URL and any other web sites that may be mentioned. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise) or for any purpose, without the express written permission of Sesame Technology.

Sesame Technology may have patents, patent applications, trademarks, trademarks applied for, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Sesame Technology, the furnishing of this document does not give you any license to these patents, trademarks, copyrights or other intellectual property.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

ExtraView Database Tables

Table_Name	Table_Description
ALLOWED_VALUES	A list of values allowed for a given Allowed_Value_Type.
ALLOWED_VALUE_TYPES	A list of field types with Allowed_Values.
APPLICATION_DEFAULT	System control and configuration for ExtraView. Each record contains a name of a system control variable and the value itself that controls a system-wide variable.
AREA	Division or department within an organization. The entire enterprise is composed of areas, and at a lower level, of projects. This is used with the PROJECT table to create a hierarchy with inheritance for such features as field-level security. Area "0" and project "0" control the top-level default behavior. Project "1" of Area "0" can be defined to have overriding behaviors for Area "0", Project "0", but would otherwise inherit all behaviors.
ATTACHMENT	Metadata about an external attachment, which is any MIME-compliant file associated with an issue. Zero-to-many attachments are related to an issue (the PROBLEM table). One ATTACHMENT_CONTENT is related to an ATTACHMENT.
ATTACHMENT_CONTENT	Binary content of an external attachment, which is any MIME-compliant file associated with an issue through the ATTACHMENT.
CATEGORY	The major classification of an issue by business purpose. This table contains metadata about all possible states for Category.
CUSTOMER	Contains basic information about the customers associated with issues.
DATA_DICTIONARY	The repository and description for meta data for all fields and user-definable screen objects, including field-level help.

DB_COOKIE_DATA	Name-value pair data used for maintaining transaction state. A screen dialog will have a set of records that store all the relevant information, so that the dialog can continue without storing all the information in the screen itself.
FILES	Contains metadata about external attachments stored using the Oracle Application Server technique. This has been replaced by the ATTACHMENT, ATTACHMENT_CONTENT tables for new implementations.
INTEREST_LIST	A table that contains a list of users who are interested in receiving emails about an issue or who is a member of an interest list defined on a field within the system.
LAYOUT	Contains information about the design of a screen or report. Used by the Layout Editor to create flexible screen and report layouts.
LAYOUT_ELEMENT	Contains information about individual screen or report fields in a layout. May be a field, a label or a button.
LAYOUT_ELEMENT_ATTRIBUTE	Contains specific attribute information about individual elements within a layout. For example, dependencies that control the visibility of a layout field are stored here.
LAYOUT_TYPE	Major classification of layouts by screen or report function within ExtraView.
MODULE	A repeating list of components that typically belong to a product and is associated with an issue.
MODULE_TYPE	Major classification of modules across all products.
PRIORITY	The relative priority assigned to the resolution of an issue.
PRIVACY_GROUP	A group of users who are given access to problems associated with that group name.
PRIVACY_USER	A list of users who belong to a privacy group.

PROBLEM	Contains information about an issue, problem, case, or trouble ticket. This is the core entity in ExtraView.
PROBLEM_CASE	Intersection data associating an issue with a particular customer, including the identifier from an external CRM system.
PROBLEM_GROUP	Intersection data associating multiple, related issues for reporting and resolution. Generally, if two issues are opened for different customers concerning the same thing, or a new issue is opened when proper research would have led the person to review an old issue, this provides a means of associating and tracking them together.
PROBLEM_HIST	Contains a history of new, changed, or deleted issues. Includes the fields from the PROBLEM table after the change, or before the delete.
PROBLEM_MODULE	Intersection data associating issues to modules, including status and to whom it is assigned.
PROBLEM_MODULE_HIST	A complete record of history information for the PROBLEM_MODULE.
PROBLEM_RELEASE	Intersection data associating issues to product releases, including status and to whom it is assigned.
PROBLEM_RELEASE_HIST	A complete record of history information for the PROBLEM_RELEASE.
PROBLEM_TEXT	Contains long text associated with an issue. The table is constructed so that each record can contain up to 32k of data.
PROBLEM_TEXT_HIST	A complete record of all changes to PROBLEM_TEXT.
PROBLEM_UDF	Intersection data associating issues to User Defined Fields, including the text, numeric, or date value.
PROBLEM_UDF_HIST	A complete record of all changes to PROBLEM_UDF.

PRODUCT	Contains information about a given product. Products typically contain releases and modules, and are associated with issues.
PRODUCT_LINE	Contains information about a plurality of related products.
PRODUCT_PRODUCT_LINE	Intersection data associating products to a product line.
PRODUCT_RELEASE	Contains information about a named release of a given product.
PROJECT	An individual project within an Area. The entire enterprise is composed of areas, and at a lower level, of projects. This is used with the PROJECT table to create a hierarchy with inheritance for such features as field-level security. Area "0" and project "0" control the top-level default behavior. Project "1" of Area "0" could be defined as have overriding behaviors for Area "0", Project "0", but would otherwise inherit all behaviors.
REPORT_COLUMNS	The columns which are to appear within a given system or user-defined report.
REPORT_SORT	The sort order for a system or user-defined report.
RESOLUTION	The list of potential resolutions for a problem.
SECURITY_GROUP	The major grouping of ExtraView users into functional security areas.
SECURITY_GROUP_USER	A list of the users belonging to a Security Group.
SECURITY_MODULE	A single function within ExtraView that can be viewed or updated.
SECURITY_PERMISSION	A list of read and write permissions to Security Modules for a Security Group.
SECURITY_USER	An ExtraView user. This table contains all the data associated with a user, including their personal preferences.
SEVERITY_LEVEL	The list of possible severity levels for a problem.

STATUS	The list of possible statuses for a problem.
STATUS_RULE	A matrix of legal statuses to which a problem can be changed, given its current status. This can be keyed on areas, project and user group, or product.
SYSTEM_LOG	Information about various system events performed by users, such as Sign On attempts and system configuration changes.
SYSTEM_LOG_TYPE	A list of event types that can be logged into the System Log.
UDF	The description of all User Defined Fields in the system.
UDF_LIST	The possible values for a given List type User Defined Field.
USER_GLOBAL	Information for a user that overrides system default settings.
USER_SESSION	Information for a particular session for a given user, used to track Sign-Ons.

ALLOWED_VALUES

A list of values allowed for a given Allowed_Value_Type.

Column Name	Title	Format	Text
CHILD_VALUE		VARCHAR2(256)	Allowed value.
OWNER		VARCHAR2(30)	Owner of this allowed value.
PARENT_VALUE		VARCHAR2(256)	Parent of allowed value.
SORT_SEQ		VARCHAR2(256)	The sequence in which the allowed values, belonging to a certain allowed value type, are sorted when displayed.
TYPE_ID		NUMBER	The identifier of the allowed value type to which this allowed value belongs.

ALLOWED_VALUE_TYPE

This table contains a list of field types with Allowed_Values.

Column Name	Title	Format	Text
CHILD_NAME		VARCHAR2(256)	A list of field types with ALLOWED_VALUES.
PARENT_NAME		VARCHAR2(256)	Allowed value type.
TITLE		VARCHAR2(40)	Parent of allowed value type.
TYPE_ID		NUMBER	External name of this allowed value type.

APPLICATION_DEFAULT

This table provides for ExtraView system control and configuration. Each record contains a name of a system control variable and the value itself that controls a system-wide variable.

Column Name	Title	Format	Text
COMMENTS		VARCHAR2(64)	Description of this application default.
NAME		VARCHAR2(30)	The name of this application default. A record is uniquely identified by its name.
VALUE		VARCHAR2(4000)	The value of this application default.

AREA

This table refers to the division or department within an organization. The entire enterprise is composed of areas, and at a lower level, of projects. This is used with the PROJECT table to create a hierarchy with inheritance for such features as field-level security. Area "0" and project "0" control the top-level default behavior. Project "1" of Area "0" can be defined to have overriding behaviors for Area "0", Project "0", but would otherwise inherit all behaviors.

Column Name	Title	Format	Text
AREA_ID		NUMBER	A unique internal key for this area.
CREATED_BY_USER		VARCHAR2(30)	The user who created this area.
DATE_CREATED		DATE	The date when this area was created.
LAST_DATE_UPDATED		DATE	The date when this area was last updated.
LAST_UPDATED_BY_USER		VARCHAR2(30)	The user that last updated this area.
TITLE		VARCHAR2(40)	External name of this area.

ATTACHMENT

Metadata about an external attachment, which is any MIME-compliant file associated with an issue. Zero-to-many attachments are related to an issue (the PROBLEM table). One ATTACHMENT_CONTENT is related to an ATTACHMENT.

Column Name	Title	Format	Text
ATTACHMENT_ID		NUMBER	A unique internal key for this attachment.
CONTENT_TYPE		VARCHAR2(100)	The type of file in this attachment.
CREATED_BY_USER		VARCHAR2(30)	The user who created this attachment.
DATE_CREATED		DATE	The date when this attachment was created.
FILE_DESC		VARCHAR2(1000)	Description of the file in this attachment.
FILE_NAME		VARCHAR2(256)	Name of the file in this attachment.
FILE_SIZE		NUMBER	Size in bytes of the file in this attachment.
PATH		VARCHAR2(1000)	Path of the directory from which this attachment was uploaded.
PROBLEM_ID		NUMBER	The identifier of the issue to which this attachment belongs.

ATTACHMENT_CONTENT

Binary content of an external attachment, which is any MIME-compliant file associated with an issue through the ATTACHMENT.

Column Name	Title	Format	Text
ATTACHMENT_ID		NUMBER	The identifier of the attachment to which this attachment content belongs.
CONTENT_BLOB		BLOB(4000)	The content of the attachment.

CATEGORY

Category refers to the major classification of an issue by its business purpose. This table contains metadata about all possible states for a given Category.

Column Name	Title	Format	Text
CATEGORY_ID		NUMBER	A unique internal key for this category.
CREATED_BY_USER		VARCHAR2(30)	The user who created this category.
DATE_CREATED		DATE	The date when this category was created.
LAST_DATE_UPDATED		DATE	The date when this category was last updated.
LAST_UPDATED_BY_USER		VARCHAR2(30)	The user that last updated this category.
NAME		VARCHAR2(30)	The name of this category. A record is uniquely identified by its name.
TITLE		VARCHAR2(40)	External name of this category.

CUSTOMER

Contains basic information about the customers associated with issues.

Column Name	Title	Format	Text
CRM_CUSTOMER_CODE		VARCHAR2(30)	Reference to the customer code in a CRM system to which Extraview has an interface.
CUSTOMER_ID		NUMBER	A unique internal key for this customer.
CUSTOMER_NAME		VARCHAR2(256)	Name of the customer.
SOUNDEX_CUSTOMER_NAME		VARCHAR2(4)	Soundex name of the customer.

DATA_DICTIONARY

The repository and description for meta data for all fields and user-definable screen objects, including field-level help.

Column Name	Title	Format	Text
ALLOW_ADD_TO_LIST		VARCHAR2(1)	A flag indicating that the product is or is not allowed in a list.
COLUMN_NAME		VARCHAR2(30)	The column name where the data is stored inside the table defined in TABLE_NAME.
CREATED_BY_USER		VARCHAR2(30)	The user who created this data dictionary entry.
DATE_CREATED		DATE	The date when this data dictionary entry was created.
DEFAULT_FORMAT		VARCHAR2(40)	The default format for display of the item. Not currently used.
DEFAULT_SIZE		NUMBER	The default size, in characters, for display of the item. Not currently used.
DEFAULT_VALUE		VARCHAR2(100)	The default value that will populate the field in the mode where a new issue is being added. This can be overwritten to remember the last value of the field for a specific user.

DEPENDENT_SQL		VARCHAR2(4000)	
DISPLAY_AS_URL		VARCHAR2(1)	Y or N to signify that the field should be displayed as a link.
DISPLAY_FORMAT		VARCHAR2(10)	The default display format for the entry.
DISPLAY_TYPE		VARCHAR2(10)	The display type of the entry. This controls the appearance of the entry as it is displayed on any screen or report.
ENABLE_INTEREST_LIST		VARCHAR2(1)	Indicates whether this data dictionary entry will be available for an interest list.
HELP_TEXT		VARCHAR2(2000)	Tool tips that appear when the mouse cursor is placed over an entry.
HELP_URL		VARCHAR2(100)	Reference to the online help for this data dictionary item.
LAST_DATE_UPDATED		DATE	The date that this data dictionary entry was last updated.
LAST_UPDATED_BY_USER		VARCHAR2(30)	The user that last updated this data dictionary entry.

NAME		VARCHAR2(30)	The name of this data dictionary value. A record is uniquely identified by its name.
ORDER_BY_SQL		VARCHAR2(500)	
PARENT1_FIELD_NAME		VARCHAR2(30)	
PARENT1_SQL		VARCHAR2(4000)	
PARENT2_FIELD_NAME		VARCHAR2(30)	
PARENT2_SQL		VARCHAR2(4000)	
PRIMARY_SQL		VARCHAR2(4000)	
REQUIRED_FIELD		VARCHAR2(1)	Y or N to signify whether the field is required. Now superseded by a layout attribute.
SAVE_LAST_VALUE		VARCHAR2(1)	Y or N. Enables the saving of the last value set by a user for future entry into the add issue form.
SELECT_FOR_REPORTS		VARCHAR2(1)	A flag indicating if this data dictionary entry may be displayed on a report.
TABLE_NAME		VARCHAR2(30)	The table name where the data is stored for the data dictionary entry. Used along with COLUMN_NAME.
TITLE		VARCHAR2(40)	External name of this data dictionary item.

TYPE		VARCHAR2(30)	Type of data dictionary item. Allowed values are: DATABASE, LABEL, SCREEN and UDF.
URL		VARCHAR2(255)	The URL that is used as a link from the field to another location on the network.

DB_COOKIE_DATA

Name-value pair data used for maintaining transaction state. A screen dialog will have a set of records that store all the relevant information, so that the dialog can continue without storing all the information in the screen itself.

Column Name	Title	Format	Text
CKEY		NUMBER	Session identifier for the cookie. The session is generally a screen dialog which completes a transaction. The CKEY together with the CNAME uniquely identifies a record.
CNAME		VARCHAR2(40)	Name of the cookie. The CKEY together with the CNAME uniquely identifies a record.
CVALUE		VARCHAR2(4000)	Value of the cookie.
CVALUE_LONG		LONG	Long value of the cookie.

ENCLOSURE

Column Name	Title	Format	Text
CREATED_BY_USER		VARCHAR2(30)	The user who created this enclosure.
DATE_CREATED		DATE	The date when this enclosure was created.
ENCLOSURE_ID		NUMBER	A unique internal key for this enclosure.
ENCLOSURE_TEXT		LONG	The text of this enclosure.
ENCLOSURE_TYPE_NAME		VARCHAR2(30)	The identifier of the enclosure type to which this enclosure belongs.
LAST_DATE_UPDATED		DATE	The date when this enclosure was last updated.
LAST_UPDATED_BY_USER		VARCHAR2(30)	The user that last updated this enclosure.
PROBLEM_ID		NUMBER	The identifier of the issue to which this enclosure belongs.
TITLE		VARCHAR2(80)	External name of this enclosure.

ENCLOSURE_TYPE

Column Name	Title	Format	Text
CREATED_BY_USER		VARCHAR2(30)	The user who created this enclosure type.
DATE_CREATED		DATE	The date that this enclosure type was created.
LAST_DATE_UPDATED		DATE	The date that this enclosure type was last updated.
LAST_UPDATED_BY_USER		VARCHAR2(30)	The user that last updated this enclosure type.
NAME		VARCHAR2(30)	The name of this enclosure type. A record is uniquely identified by its name.
TITLE		VARCHAR2(40)	External name of this enclosure type.

FILES

Contains metadata about external attachments stored using the Oracle Application Server technique. This has been replaced by the ATTACHMENT, ATTACHMENT_CONTENT tables for new implementations.

Column Name	Title	Format	Text
DESCRIPTION		VARCHAR2(1000)	Description of the file attachment.
FILE_LENGTH		NUMBER	The length of the file to be uploaded as an attachment.
FILE_NAME		VARCHAR2(2000)	The internal name of the file to be uploaded as an attachment. The file name is of the format: <issue number>_<sequence number within the issue>_<the name of the file to be uploaded, including suffix>.
FILE_TYPE		VARCHAR2(30)	The MIME type of the file to be uploaded as an attachment. The file type must be one of the following: doc, htm, jpg, ppt, sql, txt, xls.
ID		NUMBER	The identifier of the issue to which this file belongs.
OID		NUMBER	A unique internal key for this file that is generated by the Oracle Application Server.
SHORT_FILE		VARCHAR2(100)	The name of the file to be uploaded, including suffix.

TIMESTAMP		DATE	The date and time the file was entered or last updated.
USER_NAME		VARCHAR2(30)	The name of the user that uploaded this file.

INTEREST_LIST

A table that contains a list of users who are interested in receiving emails about an issue or who is a member of an interest list defined on a field within the system.

Column Name	Title	Format	Text
ID		NUMBER	A unique internal key for this interest list.
SECURITY_USER_ID		VARCHAR2(30)	The identifier of the security user to which this interest list belongs.
TYPE_ID		VARCHAR2(40)	The identifier of the data dictionary name which this interest list references.

LAYOUT

Contains information about the design of a screen or report. Used by the Layout Editor to create flexible screen and report layouts.

Column Name	Title	Format	Text
AREA_ID		NUMBER	The identifier of the area to which this layout belongs.
CREATED_BY_USER		VARCHAR2(30)	The user who created this layout.
DATE_CREATED		DATE	The date when this layout was created.
DESCRIPTION		VARCHAR2(100)	Description of screen or report layout.
GRID_COLS		NUMBER	Maximum number of columns for this layout.
GRID_ROWS		NUMBER	Maximum number of rows for this layout.
LAST_DATE_UPDATED		DATE	The date when this layout was last updated.
LAST_UPDATED_BY_USER		VARCHAR2(30)	The user that last updated this layout.
LAYOUT_ID		NUMBER	A unique internal key for this layout.
PROJECT_ID		NUMBER	The identifier of the project to which this layout belongs.

SECURITY_GROUP_ID		VARCHAR2(18)	The identifier of the security group to which this layout belongs.
TEXT		VARCHAR2(4000)	
TITLE		VARCHAR2(40)	External name of this layout.
TYPE_ID		NUMBER	The identifier of the layout type to which this layout belongs.

LAYOUT_ELEMENT

Contains information about individual screen or report fields in a layout. May be a field, a label or a button.

Column Name	Title	Format	Text
COL_NUM		NUMBER	Horizontal position of layout element.
COL_SPAN		NUMBER	Horizontal span of layout element.
CREATED_BY_USER		VARCHAR2(30)	The user who created this layout element.
DATE_CREATED		DATE	The date when this layout element was created.
DD_NAME		VARCHAR2(30)	The identifier of the data dictionary element to which this layout element belongs.
ELEMENT_ID		NUMBER	A unique internal key for this layout element.
LABEL_FLAG		VARCHAR2(1)	Indicates whether label must be located in standard the position next to the associated field on a screen.

LAST_DATE_UPDATED		DATE	The date when this layout element was last updated.
LAST_UPDATED_BY_USER		VARCHAR2(30)	The user that last updated this layout element.
LAYOUT_ID		NUMBER	The identifier of the layout to which this layout element belongs.
REQUIRED_FLAG		CHAR(1)	A flag indicating that the layout element is or is not required.
ROW_NUM		NUMBER	Vertical position of layout element.
ROW_SPAN		NUMBER	Vertical span of layout element.
TEXT		VARCHAR2(4000)	

LAYOUT_ELEMENT_ATTRIBUTE

Contains specific attribute information about individual elements within a layout. For example, dependencies that control the visibility of a layout field are stored here.

Column Name	Title	Format	Text
ATTRIBUTE_NAME		VARCHAR2(40)	The name of this layout element attribute. The attribute_name together with the ELEMENT_ID uniquely identifies a record.
ATTRIBUTE_VALUE		VARCHAR2(4000)	The value of this layout element attribute.
CREATED_BY_USER		VARCHAR2(30)	The user who created this layout element attribute.
DATE_CREATED		DATE	The date when this layout element attribute was created.
ELEMENT_ID		NUMBER	The identifier of the layout element to which this layout element attribute belongs.
LAST_DATE_UPDATED		DATE	The date when this layout element attribute was last updated.

LAST_UPDATED_BY_USER		VARCHAR2(30)	The user that last updated this layout element attribute.
----------------------	--	--------------	---

LAYOUT_TYPE

Major classification of layouts by screen or report function within ExtraView.

Column Name	Title	Format	Text
CREATED_BY_USER		VARCHAR2(30)	The user who created this layout type.
DATE_CREATED		DATE	The date when this layout type was created.
LAST_DATE_UPDATED		DATE	The date when this layout type was last updated.
LAST_UPDATED_BY_USER		VARCHAR2(30)	The user that last updated this layout type.
NAME		VARCHAR2(30)	Internal name of this layout type.
TITLE		VARCHAR2(40)	External name of this layout type.
TYPE_ID		NUMBER	A unique internal key for this layout type.
CREATED_BY_USER		VARCHAR2(30)	The user who created this layout type.
DATE_CREATED		DATE	The date when this layout type was created.
LAST_DATE_UPDATED		DATE	The date when this layout type was last updated.
LAST_UPDATED_BY_USER		VARCHAR2(30)	The user that last updated this layout type.

NAME		VARCHAR2(30)	Internal name of this layout type.
TITLE		VARCHAR2(40)	External name of this layout type.
TYPE_ID		NUMBER	A unique internal key for this layout type.
CREATED_BY_USER		VARCHAR2(30)	The user who created this layout type.
DATE_CREATED		DATE	The date when this layout type was created.
LAST_DATE_UPDATED		DATE	The date when this layout type was last updated.
LAST_UPDATED_BY_USER		VARCHAR2(30)	The user that last updated this layout type.
NAME		VARCHAR2(30)	Internal name of this layout type.
TITLE		VARCHAR2(40)	External name of this layout type.
TYPE_ID		NUMBER	A unique internal key for this layout type.

LOOKUP

Column Name Title Format Text

LOOKUP_ID		NUMBER	A unique internal key for this lookup.
LOOKUP_TYPE_ID		NUMBER	The identifier of the lookup type to which this lookup belongs.
SORT_SEQUENCE		NUMBER	
TITLE		VARCHAR2(40)	External name of this lookup.

LOOKUP_TYPE

Column Name	Title	Format	Text
AREA		VARCHAR2(30)	The user who created this lookup type.
CREATED_BY_USER		DATE	The date when this lookup type was created.
DATE_CREATED		VARCHAR2(80)	Description of lookup type.
DESCRIPTION		DATE	The date when this lookup type was last updated.
LAST_DATE_UPDATED		VARCHAR2(30)	The user that last updated this lookup type.
LAST_UPDATED_BY_USER		NUMBER	A unique internal key for this lookup type.
LOOKUP_TYPE_ID		NUMBER	The identifier of the project to which this lookup type belongs.
PROJECT		VARCHAR2(30)	Type of sort performed on this lookup type, e.g., numeric or alphabetic.
SORT_TYPE		VARCHAR2(40)	External name of this lookup type.
TITLE		VARCHAR2(30)	The user who created this lookup type.

MODULE

A repeating list of components that typically belong to a product and is associated with an issue.

Column Name	Title	Format	Text
CREATED_BY_USER		VARCHAR2(30)	The user who created this module.
DATE_CREATED		DATE	The date when this module was created.
DEFAULT_OWNER	Module Owner	VARCHAR2(30)	The default owner of the module, used to auto-assign issues.
LAST_DATE_UPDATED		DATE	The date when this module was last updated.
LAST_UPDATED_BY_USER		VARCHAR2(30)	The user that last updated this module.
MODULE_ID		NUMBER	A unique internal key for this module.
MODULE_TYPE	Module Type	VARCHAR2(12)	The identifier of the module type to which this module belongs.
NAME	Module	VARCHAR2(40)	A module is a significant part of a product.
PRODUCT_NAME		VARCHAR2(30)	The identifier of the product to which this module belongs.
TITLE		VARCHAR2(100)	External name of this module.

MODULE_TYPE

The major classifications of modules across all products.

Column Name	Title	Format	Text
CREATED_BY_USER		VARCHAR2(30)	The user who created this module type.
DATE_CREATED		DATE	The date when this module type was created.
LAST_DATE_UPDATED		DATE	The date when this module type was last updated.
LAST_UPDATED_BY_USER		VARCHAR2(30)	The user that last updated this module type.
NAME		VARCHAR2(12)	The name of this module. A record is uniquely identified by its name.
TITLE		VARCHAR2(30)	External name of this module type.
LAST_UPDATED_BY_USER		VARCHAR2(30)	The user that last updated this priority.
NAME		VARCHAR2(15)	Internal name of this priority.
PRIORITY_ID		NUMBER	A unique internal key for this priority.
TITLE		VARCHAR2(30)	External name of this priority.

PRIORITY

The relative priority assigned to the resolution of an issue.

Column Name	Title	Format	Text
CREATED_BY_USER		VARCHAR2(30)	The user who created this priority.
DATE_CREATED		DATE	The date when this priority was created.
LAST_DATE_UPDATED		DATE	The date when this priority was last updated.

PRIVACY_GROUP

A group of users who are given access to problems associated with that group name.

Column Name	Title	Format	Text
PRIVACY_GROUP_NAME		VARCHAR2(30)	Internal name of this privacy group.
PRIVACY_GROUP_TITLE		VARCHAR2(80)	External name of this privacy group.

PRIVACY_USER

A list of users who belong to a privacy group.

Column Name	Title	Format	Text
PRIVACY_GROUP_NAME		VARCHAR2(30)	The identifier of the Privacy Group to which this privacy user belongs.
SECURITY_USER_ID		VARCHAR2(30)	The identifier of the security user to which this privacy user belongs.

PROBLEM

Contains information about an issue, problem, case, or trouble ticket. This is the core entity in ExtraView.

Column Name	Title	Format	Text
ALT_ID	Alt ID	VARCHAR2(30)	Special format issue identifier or the legacy ID from a system from which data has been converted into ExtraView.
AREA_ID	Area	NUMBER	The business area to which this problem belongs.
ASSIGNED_TO	Assigned To	VARCHAR2(30)	The security ID of the user to which this issue is currently assigned.
CATEGORY	Category	VARCHAR2(30)	The category of the problem.
CONTACT	Contact	VARCHAR2(30)	The contact person for this issue.
DATE_CLOSED		DATE	The date when this issue was closed.
DATE_CREATED		DATE	The date when this issue was created.

DATE_LAST_STATUS_CHANGE		DATE	The date when an issue last changed status.
HOURS_BILLED		NUMBER	Number of hours that were billed to this issue.
ID	Problem #	VARCHAR2(30)	This is the key field by which new issues are filed. The number is allocated automatically as the problem is added. Issues may be referred to by this number.
LAST_CHANGE_USER	Changed by	VARCHAR2(30)	The last person who changed this issue.
ORIGINATOR	Originator	VARCHAR2(30)	The originator of the issue.
OWNER	Owner	VARCHAR2(15)	The owner of this issue.
PRIORITY	Priority	VARCHAR2(15)	The priority of the issue.
PRIVACY	View	VARCHAR2(30)	This field determines whether the issue is to be displayed outside the company or privacy group.

PRODUCT_LINE	Product Line	VARCHAR2(30)	The name of the product line. A product can belong to one or more product lines.
PRODUCT_NAME	Product	VARCHAR2(30)	The identifier of the product to which this issue belongs.
PROJECT_ID	Project	NUMBER	The identifier of the project to which this issue belongs.
RELEASE_FIXED		VARCHAR2(50)	The release where this issue was fixed.
RELEASE_FOUND		VARCHAR2(50)	The release where this issue was found.
RESOLUTION	Disposition	VARCHAR2(30)	The identifier of the resolution to which this problem belongs.
SEVERITY_LEVEL	Severity	VARCHAR2(15)	The severity of this issue.
SHORT_DESCR	Title	VARCHAR2(255)	The title of the problem, as it will appear on a report.
STATUS	Status	VARCHAR2(15)	The status of this issue.

PROBLEM_CASE

Intersection data associating an issue with a particular customer, including the identifier from an external CRM system.

Column Name	Title	Format	Text
CRM_CASE_ID		VARCHAR2(50)	Reference to the case identifier in a CRM system to which ExtraVew has an interface.
CUSTOMER_ID		NUMBER	The identifier of the customer to which this problem belongs in an external CRM system.
PROBLEM_ID		NUMBER	The identifier of the issue to which this problem case belongs.

PROBLEM_GROUP

Intersection data associating multiple, related issues for reporting and resolution. Generally, if two issues are opened for different customers concerning the same thing, or a new issue is opened when proper research would have led the person to review an old issue, this provides a means of associating and tracking them together.

Column Name	Title	Format	Text
CREATED_BY_USER		VARCHAR2(30)	The user who created this problem group.
DATE_CREATED		DATE	The date when this problem group was created.
LAST_DATE_UPDATED		DATE	The date when this problem group was last updated.
LAST_UPDATED_BY_USER		VARCHAR2(30)	The user that last updated this problem group.
PROBLEM_GROUP_ID		NUMBER	A numeric key. The PROBLEM_GROUP_ID together with the PROBLEM_ID uniquely identifies a record.
PROBLEM_ID		NUMBER	The identifier of the issue to which this problem group belongs.

PROBLEM_HIST

Contains a history of new, changed, or deleted issues. Includes the fields from the PROBLEM table after the change, or before the delete.

Column Name	Title	Format	Text
ALT_ID		NUMBER	Special format issue identifier or the legacy id from a system from which data has been converted into ExtraView.
AREA_ID		VARCHAR2(30)	The identifier of the area to which this issue belongs.
ASSIGNED_TO		VARCHAR2(30)	The security ID of the user to which this issue is currently assigned.
CATEGORY		VARCHAR2(30)	The category of the problem.
CHANGE_TYPE		CHAR(1)	Type of change: insert, update or delete (I, U or D).
CONTACT		VARCHAR2(30)	The contact person for this issue.
DATE_CLOSED		DATE	The date when this issue was closed.

DATE_CREATED		DATE	The date when this issue was created.
DATE_LAST_STATUS_CHANGE		DATE	The date when an issue last changed status.
HIST_TIMESTAMP		DATE	The date and time when the issue changed status.
HOURS_BILLED		NUMBER	Number of hours that were billed to this issue.
ID		NUMBER	This is the key field by which new issues are filed. The number is allocated automatically as the problem is added. Issues may be referred to by this number.
LAST_CHANGE_USER		VARCHAR2(30)	The last person who changed this issue.
ORIGINATOR		VARCHAR2(30)	The originator of the issue.
OWNER		VARCHAR2(30)	The owner of this issue.
PRIORITY		VARCHAR2(15)	The priority of the issue.

PRIVACY		VARCHAR2(15)	This field determines if the issue is to be displayed outside the company of privacy group. The most common alternatives are: PUBLIC or PRIVATE.
PRODUCT_LINE		VARCHAR2(30)	The name of the product line. A product can belong to one or more product lines.
PRODUCT_NAME		VARCHAR2(30)	The identifier of the product to which this issue belongs.
PROJECT_ID		NUMBER	The identifier of the project to which this issue belongs.
RELEASE_FIXED		VARCHAR2(50)	The release where this issue was fixed.
RELEASE_FOUND		VARCHAR2(50)	The release where this issue was found.
RESOLUTION		VARCHAR2(30)	The identifier of the resolution to which this problem belongs.

SEVERITY_LEVEL		VARCHAR2(15)	The severity of this issue.
SHORT_DESCR		VARCHAR2(255)	The title of the problem, as it will appear on a report.
STATUS		VARCHAR2(15)	The status of this issue.
TIMESTAMP		DATE	The date and time the issue was entered or last updated.

PROBLEM_MODULE

Intersection data associating issues to modules, including status and to whom it is assigned.

Column Name	Title	Format	Text
ASSIGNED_TO	Assigned	VARCHAR2(30)	The security ID of the user to which this module is currently assigned.
ID		NUMBER	The identifier of the issue to which this problem module belongs.
LAST_CHANGE_USER		VARCHAR2(30)	The last person who changed this problem module.
MODULE_ID	Module	NUMBER	The identifier of the module to which this problem module belongs.
PROBLEM_MODULE_ID		NUMBER	The identifier of the module to which this problem module belongs.
RC_VERSION	Version	NUMBER	The version of the module within which the issue exists.
STATUS	Module Status	VARCHAR2(15)	The status of this problem for a module.
TIMESTAMP		DATE	The date and time the problem module was entered or last updated.

PROBLEM_MODULE_HIST

This table contains a complete record of history information for the PROBLEM_MODULE.

Column Name	Title	Format	Text
ASSIGNED_TO		VARCHAR2(30)	The security ID of the user to which this module is currently assigned.
CHANGE_TYPE		CHAR(1)	Type of change: insert, update or delete (I, U or D).
HIST_TIMESTAMP		DATE	The date and time when the problem module changed status.
ID		NUMBER	The identifier of the issue to which this problem module belongs.
LAST_CHANGE_USER		VARCHAR2(30)	The last person who changed this problem module.
MODULE_ID		NUMBER	The identifier of the module to which this problem module belongs.
PROBLEM_MODULE_ID		NUMBER	The identifier of the module to which this problem module belongs.
RC_VERSION		NUMBER	The version of the module within which the issue exists.
STATUS		VARCHAR2(15)	The status of this problem for a module.

TIMESTAMP		DATE	The date and time the problem module was entered or last updated.

PROBLEM_RELEASE

Intersection data associating issues to product releases, including status and to whom it is assigned.

Column Name	Title	Format	Text
ASSIGNED_TO	Release Assigned To	VARCHAR2(30)	The security ID of the person to which this problem release is assigned.
CREATED_BY_USER		VARCHAR2(30)	The user who created this problem release.
DATE_CREATED		DATE	The date when this problem release was created.
LAST_DATE_UPDATED		DATE	The date when this problem release was last updated.
LAST_UPDATED_BY_USER		VARCHAR2(30)	The user that last updated this problem release.
OWNER	Release Owner	VARCHAR2(30)	The owner of a particular release.
PRIORITY	Priority	VARCHAR2(15)	The priority of the release.
PROBLEM_ID		NUMBER	The identifier of the issue to which this problem release belongs.
PROBLEM_RELEASE_ID		NUMBER	A unique internal key for this problem release.

PRODUCT_NAME		VARCHAR2(30)	The identifier of the product to which this problem release belongs.
RELEASE_FIXED	Version Closed	VARCHAR2(50)	The release in which the issue was fixed.
RELEASE_FOUND	Version Open	VARCHAR2(50)	The release in which the issue was found.
RESOLUTION	Disposition	VARCHAR2(30)	The identifier of the resolution to which this problem release belongs.
SEVERITY_LEVEL	Severity	VARCHAR2(15)	The severity of a particular release.
STATUS	Release Status	VARCHAR2(15)	The status of this problem for a release.

PROBLEM_RELEASE_HIST

This table contains a complete record of history information for the PROBLEM_RELEASE.

Column Name	Title	Format	Text
ASSIGNED_TO		VARCHAR2(30)	The security ID of the person to which this problem release is assigned.
CHANGE_TYPE		CHAR(1)	Type of change: insert, update or delete (I, U or D).
CREATED_BY_USER		VARCHAR2(30)	The user who created this problem release.
DATE_CREATED		DATE	The date when this problem release was created.
HIST_TIMESTAMP		DATE	The date and time when the problem release changed status.
LAST_DATE_UPDATED		DATE	The date when this problem release was last updated.
LAST_UPDATED_BY_USER		VARCHAR2(30)	The user that last updated this problem release.
OWNER		VARCHAR2(30)	The owner of a particular release.
PRIORITY		VARCHAR2(15)	The priority of the release.

PROBLEM_ID		NUMBER	The identifier of the issue to which this problem release belongs.
PROBLEM_RELEASE_ID		NUMBER	A unique internal key for this problem release.
PRODUCT_NAME		VARCHAR2(30)	The identifier of the product to which this problem release belongs.
RELEASE_FIXED		VARCHAR2(50)	The release in which the issue was fixed.
RELEASE_FOUND		VARCHAR2(50)	The release in which the issue was found.
RESOLUTION		VARCHAR2(30)	The identifier of the resolution to which this problem release belongs.
SEVERITY_LEVEL		VARCHAR2(15)	The severity of a particular release.
STATUS		VARCHAR2(15)	The status of this problem for a release.

PROBLEM_TEXT

Contains long text associated with an issue. The table is constructed so that each record can contain up to 32k of data.

Column Name	Title	Format	Text
ID_SEQ		NUMBER	The order of this text field within the issue.
PROBLEM_ID		NUMBER	The identifier of the issue to which this problem text belongs.
TEXT		VARCHAR2(4000)	The text describing this issue.
TEXT_SEQ		NUMBER	The order of this block of text within the text field.
TIMESTAMP		DATE	The date and time the problem text was entered or last updated.
UDF_ID		NUMBER	The identifier of the User Defined Field to which this problem text belongs.
USER_ID		VARCHAR2(30)	The identifier of the user to which this problem text belongs.

PROBLEM_TEXT_HIST

This table contains a complete record of all changes to PROBLEM_TEXT.

Column Name	Title	Format	Text
CHANGE_TYPE		CHAR(1)	Type of change: insert, update or delete (I, U or D).
HIST_TIMESTAMP		DATE	The date and time when the problem text changed status.
ID_SEQ		NUMBER	The order of this text field within the issue.
PROBLEM_ID		NUMBER	The identifier of the issue to which this problem text belongs.
TEXT		VARCHAR2(4000)	The text describing this issue.
TEXT_SEQ		NUMBER	The order of this block of text within the text field.
TIMESTAMP		DATE	The date and time the problem text was entered or last updated.
UDF_ID		NUMBER	The identifier of the User Defined Field to which this problem text belongs.
USER_ID		VARCHAR2(30)	The identifier of the user to which this problem text belongs.

PROBLEM_UDF

Intersection data associating issues to User Defined Fields, including text, numeric, or date value.

Column Name	Title	Format	Text
CREATED_BY_USER		VARCHAR2(30)	The user who created this problem User Defined Field.
DATE_CREATED		DATE	The date when this problem User Defined Field was created.
LAST_DATE_UPDATED		DATE	The date when this User Defined Field was last updated.
LAST_UPDATED_BY_USER		VARCHAR2(30)	The user that last updated this problem User Defined Field.
PROBLEM_ID		NUMBER	The identifier of the issue to which this problem User Defined Field belongs.
UDF_ID		NUMBER	The identifier of the User Defined Field to which this problem User Defined Field belongs.
UDF_LIST_ID		NUMBER	The identifier of the User Defined Field list to which this problem User Defined Field belongs.

VALUE		VARCHAR2(256)	The text value for this User Defined Field for an issue.
VALUE_DATE		DATE	The date value for this User Defined Field for an issue.
VALUE_NUMBER		NUMBER	The numeric value for this User Defined Field for an issue.

PROBLEM_UDF_HIST

This table contains a complete record of all changes to PROBLEM_UDF.

Column Name	Title	Format	Text
CHANGE_TYPE		CHAR(1)	Type of change: insert, update or delete (I, U or D).
CREATED_BY_USER		VARCHAR2(30)	The user who created this problem User Defined Field.
DATE_CREATED		DATE	The date when this problem User Defined Field was created.
HIST_TIMESTAMP		DATE	The date and time when the problem User Defined Field changed status.
LAST_DATE_UPDATED		DATE	The date this User Defined Field was last updated.
LAST_UPDATED_BY_USER		VARCHAR2(30)	The user that last updated this problem User Defined Field.
PROBLEM_ID		NUMBER	The identifier of the issue to which this problem User Defined Field belongs.
UDF_ID		NUMBER	The identifier of the User Defined Field to which this problem User Defined Field belongs.

UDF_LIST_ID		NUMBER	The identifier of the User Defined Field list to which this problem User Defined Field belongs.
VALUE		VARCHAR2(256)	The text value for this User Defined Field for an issue.
VALUE_DATE		DATE	The date value for this User Defined Field for an issue.
VALUE_NUMBER		NUMBER	The numeric value for this User Defined Field for an issue.

PRODUCT

This table contains information about a given product. Products typically contain releases and modules, and are associated with issues.

Column Name	Title	Format	Text
ACTIVE		VARCHAR2(1)	A flag indicating that the product is active/inactive.
CREATED_BY_USER		VARCHAR2(30)	The user who created this product.
DATE_CREATED		DATE	The date when this product was created.
EMAIL		VARCHAR2(50)	The email address that will be notified upon any changes to issues filed for this product.
LAST_DATE_UPDATED		DATE	The date when this product was last updated.
LAST_UPDATED_BY_USER		VARCHAR2(30)	The user that last updated this product.
NAME		VARCHAR2(30)	The name of this product. A record is uniquely identified by its name.
PRODUCT_ID		NUMBER	A unique internal key for this product.
SOUNDEX_PRODUCT		VARCHAR2(4)	Soundex name of the product.
TITLE		VARCHAR2(30)	External name of this product.

PRODUCT_LINE

This table contains information about a plurality of related products.

Column Name	Title	Format	Text
CREATED_BY_USER		VARCHAR2(30)	The user who created this product line.
DATE_CREATED		DATE	The date when this product line was created.
LAST_DATE_UPDATED		DATE	The date when this product line was last updated.
LAST_UPDATED_BY_USER		VARCHAR2(30)	The user that last updated this product line.
NAME		VARCHAR2(30)	The name of this product line. A record is uniquely identified by its name.
PRODUCT_LINE_ID		NUMBER	A unique internal key for this product line.
TITLE		VARCHAR2(40)	External name of this product line.

PRODUCT_PRODUCT_LINE

This table contains Intersection data associating products to a product line.

Column Name	Title	Format	Text
PRODUCT_LINE_NAME		VARCHAR2(30)	The identifier of the product line to which a certain product is associated.
PRODUCT_NAME		VARCHAR2(30)	The identifier of the product to which a certain product line is associated.

PRODUCT_RELEASE

This table contains information about a named release of a given product.

Column Name	Title	Format	Text
ACTIVE	Active	VARCHAR2(1)	A flag indicating that the product release is active/inactive.
AVAILABLE_FOR_DOWNLOAD	Available For Download	VARCHAR2(1)	Indicates whether the product release is available for public download from an external customer support system.
CATEGORY		VARCHAR2(30)	The category of this product release.
CREATED_BY_USER		VARCHAR2(30)	The user who created this product release.
DATE_CODE_FREEZE	Code Freeze	DATE	The date when the code for this product release was frozen.
DATE_CREATED		DATE	The date when this product release was created.
DATE_FIRST_CUSTOMER_SHIP	First Customer Shipment	DATE	Date product was first released for customer use.

DATE_RELEASE_TO_QA	RQA	DATE	The date when this product was released to QA.
LAST_DATE_UPDATED		DATE	The date when this product release was last updated.
LAST_UPDATED_BY_USER		VARCHAR2(30)	The user that last updated this product release.
NAME	Release Code	VARCHAR2(30)	The name of this product release.
PRODUCT_NAME		VARCHAR2(30)	The identifier of the product to which this product release belongs.
RELEASE	Release Name	VARCHAR2(50)	The name of the release.
RELEASE_DIRECTORY		VARCHAR2(256)	The directory in which a software product release can be found; used for an external customer support system.
RELEASE_DOC_FILENAME	Release File	VARCHAR2(256)	Name of release notes document for external product support system.

RELEASE_TYPE	Release Type	VARCHAR2(80)	Type of release.
--------------	--------------	--------------	------------------

PROJECT

Refers to an individual project within an Area. The entire enterprise is composed of areas, and at a lower level, of projects. This is used with the PROJECT table to create a hierarchy with inheritance for such features as field-level security. Area "0" and project "0" control the top-level default behavior. Project "1" of Area "0" could be defined as have overriding behaviors for Area "0", Project "0", but would otherwise inherit all behaviors.

Column Name	Title	Format	Text
AREA_ID		NUMBER	The identifier of the area to which this project belongs.
CREATED_BY_USER		VARCHAR2(30)	The user who created this project.
DATE_CREATED		DATE	The date when this project was created.
LAST_DATE_UPDATED		DATE	The date when this project was last updated.
LAST_UPDATED_BY_USER		VARCHAR2(30)	The user that last updated this project.
PROJECT_ID		NUMBER	A unique internal key for this project.

REPORT_COLUMNS

This table contains columns which are to appear within a given system or user-defined report.

Column Name	Title	Format	Text
BREAK_FL		VARCHAR2(1)	Indicates that this named report column is used as a control break in a report.
COL_FORMAT		VARCHAR2(30)	The format of this report column.
COL_HEADING		VARCHAR2(40)	The heading of this report column.
COL_NAME		VARCHAR2(30)	Internal name of this report column.
COL_SEQUENCE		NUMBER	The sequence in which columns are displayed on a report.
NAME		VARCHAR2(60)	The identifier of the report sort order to which this report column belongs.
OWNER		VARCHAR2(30)	The identifier of the user who owns this report column.

REPORT_SORT

This table contains the sort order for a system or user-defined report.

Column Name	Title	Format	Text
CREATED_BY_USER		VARCHAR2(30)	The user who created this report sort order.
DATE_CREATED		DATE	The date when this report sort order was created.
LAST_DATE_UPDATED		DATE	The date when this report sort order was last updated.
LAST_UPDATED_BY_USER		VARCHAR2(30)	The user that last updated this report sort order.
NAME		VARCHAR2(60)	The name of this report sort order. A record is uniquely identified by its name.
TITLE		VARCHAR2(80)	External name of this report sort order.
VALUE		VARCHAR2(256)	The order of the columns after which the values in this report should be sorted.

RESOLUTION

This table contains the list of potential resolutions for a problem.

Column Name	Title	Format	Text
CREATED_BY_USER		VARCHAR2(30)	The user who created this resolution.
DATE_CREATED		DATE	The date when this resolution was created.
LAST_DATE_UPDATED		DATE	The date when this resolution was last updated.
LAST_UPDATED_BY_USER		VARCHAR2(30)	The user that last updated this resolution.
NAME		VARCHAR2(30)	The name of this resolution. A record is uniquely identified by its name.
RESOLUTION_ID		NUMBER	A unique internal key for this resolution.
TITLE		VARCHAR2(30)	The user who created this resolution.

SECURITY_GROUP

This table contains the major grouping of ExtraView users into functional security areas.

Column Name	Title	Format	Text
SECURITY_GROUP_ID		VARCHAR2(18)	A unique internal key for this security group.
SECURITY_GROUP_NAME		VARCHAR2(40)	The external name of this security group.

SECURITY_GROUP_USER

This table contains a list of the users belonging to a Security Group.

Column Name	Title	Format	Text
SECURITY_GROUP_ID		VARCHAR2(18)	The identifier of the security group to which a certain security user is associated.
SECURITY_USER_ID		VARCHAR2(30)	The identifier of the security user to which a certain security group is associated.

SECURITY_MODULE

This table contains a single function within ExtraView that can be viewed or updated.

Column Name	Title	Format	Text
DESCRIPTION		VARCHAR2(100)	Description of security module.
SECURITY_MODULE_DESC		VARCHAR2(40)	The external name of this security module.
SECURITY_MODULE_ID		VARCHAR2(50)	A unique internal key for this security module.

SECURITY_PERMISSION

This table contains a list of read and write permissions to Security Modules for a Security Group.

Column Name	Title	Format	Text
AREA_ID		NUMBER	The identifier of the area to which this security permission belongs.
PROJECT_ID		NUMBER	The identifier of the project to which this security permission belongs.
SECURITY_GROUP_ID		VARCHAR2(18)	The identifier of the security group to which a certain security module is associated.
SECURITY_MODULE_ID		VARCHAR2(50)	The identifier of the security module to which a certain security user is associated.
SECURITY_SELECT_FL		VARCHAR2(1)	A flag indicating if a certain security group has read privileges on a certain security module.
SECURITY_UPDATE_FL		VARCHAR2(1)	A flag indicating if a certain security group has update privileges on a certain security module.

SECURITY_USER

This table contains all the data associated with an ExtraView user, including their personal preferences.

Column Name	Title	Format	Text
ADDRESS_LINE1		VARCHAR2(30)	The address of this user.
ADDRESS_LINE2		VARCHAR2(30)	The address of this user.
CELL_PHONE		VARCHAR2(20)	The cell phone number of this user.
CITY		VARCHAR2(30)	The city where this user works.
COMPANY_NAME		VARCHAR2(30)	The name of the company where this user works.
COUNTRY		VARCHAR2(30)	The country where this user works.
CREATED_BY_USER		VARCHAR2(30)	The user who created this security user.
DATE_CREATED		DATE	The date when this security user was created.
DATE_FORMAT		VARCHAR2(30)	The date format chosen by this user.
DRILLDOWN_REPORT		VARCHAR2(30)	
EMAIL		VARCHAR2(50)	The email address of this user.
EMAIL_FORMAT		VARCHAR2(30)	The email format chosen by this user.

ENABLED_USER		VARCHAR2(1)	Indicates whether this user is enabled.
FAX		VARCHAR2(20)	The fax number of this user.
FIRST_NAME		VARCHAR2(25)	The first name of this user.
HOME_TELEPHONE		VARCHAR2(20)	The home telephone number of this user.
JOB_TITLE		VARCHAR2(30)	The title of this user.
LANGUAGE		VARCHAR2(30)	The language chosen by this user.
LAST_ACCESS_DATE		DATE	The last date when this user accessed the system.
LAST_DATE_UPDATED		DATE	The date when this security user entry was last updated.
LAST_NAME		VARCHAR2(25)	Last name of the user.
LAST_UPDATED_BY_USER		VARCHAR2(30)	The user that last updated this security user entry.
LOGIN_MSG_DT		DATE	The date the login message was last updated.

NOTIFY_ON_OWN_UPDATES		VARCHAR2(1)	A flag indicating if the user gets notified on his/her own updates on issues.
PAGER		VARCHAR2(20)	The pager number of this user.
PASSWORD_EXPIRY_DATE		DATE	The date when the password of this user expires.
POSTAL_CODE		VARCHAR2(10)	The postal code of this user.
RECORDS_PER_PAGE		NUMBER	The number of records that should be displayed per page for this user.
SECURITY_PASSWORD		VARCHAR2(13)	The encrypted password for this user.
SECURITY_USER_ID		VARCHAR2(30)	A unique key for this user.
SOUNDEX_LAST_NAME		VARCHAR2(4)	Soundex name of the user's last name.
STATE		VARCHAR2(20)	The state where this user works.
TIMEZONE		NUMBER	The time zone in which the user will have dates displayed.
USER_ROLE		VARCHAR2(18)	The user role of this user.

WORK_TELEPHONE		VARCHAR2(20)	The work telephone number of this user.
-----------------------	--	---------------------	--

SEVERITY_LEVEL

This table contains a list of possible severity levels for a problem.

Column Name	Title	Format	Text
CREATED_BY_USER		VARCHAR2(30)	The user that last updated this severity level.
DATE_CREATED		DATE	The user who created this severity level.
LAST_DATE_UPDATED		DATE	The date when this severity level was created.
LAST_UPDATED_BY_USER		VARCHAR2(30)	The date when this severity level was last updated.
NAME		VARCHAR2(15)	The name of this severity level. A record is uniquely identified by its name.
SEVERITY_LEVEL_ID		NUMBER	A unique internal key for this severity level.
TITLE		VARCHAR2(30)	External name of this severity level.

STATUS

This table contains a list of possible statuses for a problem.

Column Name	Title	Format	Text
CREATED_BY_USER		VARCHAR2(30)	The user who created this status.
DATE_CREATED		DATE	The date when this status was created.
LAST_DATE_UPDATED		DATE	The date when this status was last updated.
LAST_UPDATED_BY_USER		VARCHAR2(30)	The user that last updated this status.
NAME		VARCHAR2(15)	The name of this status. A record is uniquely identified by its name.
STATUS_ID		NUMBER	A unique internal key for this status.
TITLE		VARCHAR2(30)	External name of this status.

STATUS_RULE

This table contains a matrix of legal statuses to which a problem can be changed, given its current status. This can be keyed on areas, project and user group, or product.

Column Name	Title	Format	Text
-------------	-------	--------	------

AREA_ID		NUMBER	The identifier of the area to which this status rule belongs.
CREATED_BY_USER		VARCHAR2(30)	The user who created this status rule.
DATE_CREATED		DATE	The date when this status rule was created.
LAST_DATE_UPDATED		DATE	The date when this status rule was last updated.
LAST_UPDATED_BY_USER		VARCHAR2(30)	The user that last updated this status rule.
PROJECT_ID		NUMBER	The identifier of the project to which this status rule belongs.
SECURITY_GROUP_ID		VARCHAR2(18)	The identifier of the security group to which this status rule belongs.
STATUS_FROM		VARCHAR2(15)	Current status. The STATUS_FROM/STATUS_TO pair determines which status changes are allowed, given its current status.
STATUS_TO		VARCHAR2(15)	The next status. The STATUS_FROM/STATUS_TO pair determines which status changes are allowed, given its current status.
TYPE		VARCHAR2(30)	Type of status rule. Must be PRODUCT or USER or NONE.
VALUE		VARCHAR2(50)	

SYSTEM_LOG

This table contains information about various system events performed by users, such as Sign On attempts and system configuration changes.

Column Name	Title	Format	Text
LOG_DATE		DATE	The date and time when this event was recorded.
LOG_MESSAGE		VARCHAR2(1000)	A text describing the outcome of this event.
LOG_SEQ		NUMBER	A sequence number for the logged events.
LOG_TYPE		VARCHAR2(30)	The type of log. Possible log types are: SIGNON, SIGNOFF, ADD_USER_RECORD, UPDATE_USER_RECORD, ADD_PROBLEM, UPDATE_PROBLEM, DELETE_PROBLEM.
SECURITY_USER_ID		VARCHAR2(30)	The user whose action is logged.

SYSTEM_LOG_TYPE

This table contains a list of event types that can be logged into the system Log.

Column Name	Title	Format	Text
CREATED_BY_USER		VARCHAR2(30)	The user who created this system log type.
DATE_CREATED		DATE	The date when this system log type was created.
LAST_DATE_UPDATED		DATE	The date when this system log type was last updated.

LAST_UPDATED_BY_USER		VARCHAR2(30)	The user that last updated this system log type.
NAME		VARCHAR2(30)	Internal name of this system log type.
TITLE		VARCHAR2(40)	External name of this system log type.

UDF

This table contains the description of all User Defined Fields in the system.

Column Name	Title	Format	Text
AREA_ID		NUMBER	The identifier of the area to which this User Defined Field belongs.
CREATED_BY_USER		VARCHAR2(30)	The user who created this User Defined Field.
DATE_CREATED		DATE	The date when this User Defined Field was created.
HELP_TEXT		VARCHAR2(2000)	Tool tips.
HELP_URL		VARCHAR2(100)	Reference to the online help for this user-defined field.
LAST_DATE_UPDATED		DATE	The date when this User Defined Field was last updated.

LAST_UPDATED_BY_USER		VARCHAR2(30)	The user that last updated this User Defined Field.
NAME		VARCHAR2(30)	The name of this User Defined Field. A record is uniquely identified by its name.
PROJECT_ID		NUMBER	The identifier of the project to which this User Defined Field list belongs.
UDF_ID		NUMBER	A unique internal key for this User Defined Field.

UDF_LIST

This table contains the possible values for a given List type User Defined Field.

Column Name	Title	Format	Text
CREATED_BY_USER		VARCHAR2(30)	The user who created this User Defined Field list.
DATE_CREATED		DATE	The date when this User Defined Field list was created.
LAST_DATE_UPDATED		DATE	The date when this User Defined Field list was last updated.
LAST_UPDATED_BY_USER		VARCHAR2(30)	The user that last updated this User Defined Field list.
OWNER		VARCHAR2(30)	The owner of this User Defined Field list.
TITLE		VARCHAR2(40)	External name of this User Defined Field list.
UDF_ID		NUMBER	The identifier of the User Defined Field to which this list belongs.
UDF_LIST_ID		NUMBER	A unique internal key for this User Defined Field list.

USER_GLOBAL

This table contains information for a user that overrides system default settings.

Column Name	Title	Format	Text
DOMAIN		VARCHAR2(30)	
NAME		VARCHAR2(30)	The name of this user global.
SECURITY_USER_ID		VARCHAR2(30)	The identifier of the security user to which this user global entry belongs.
VALUE		VARCHAR2(4000)	

USER_SESSION

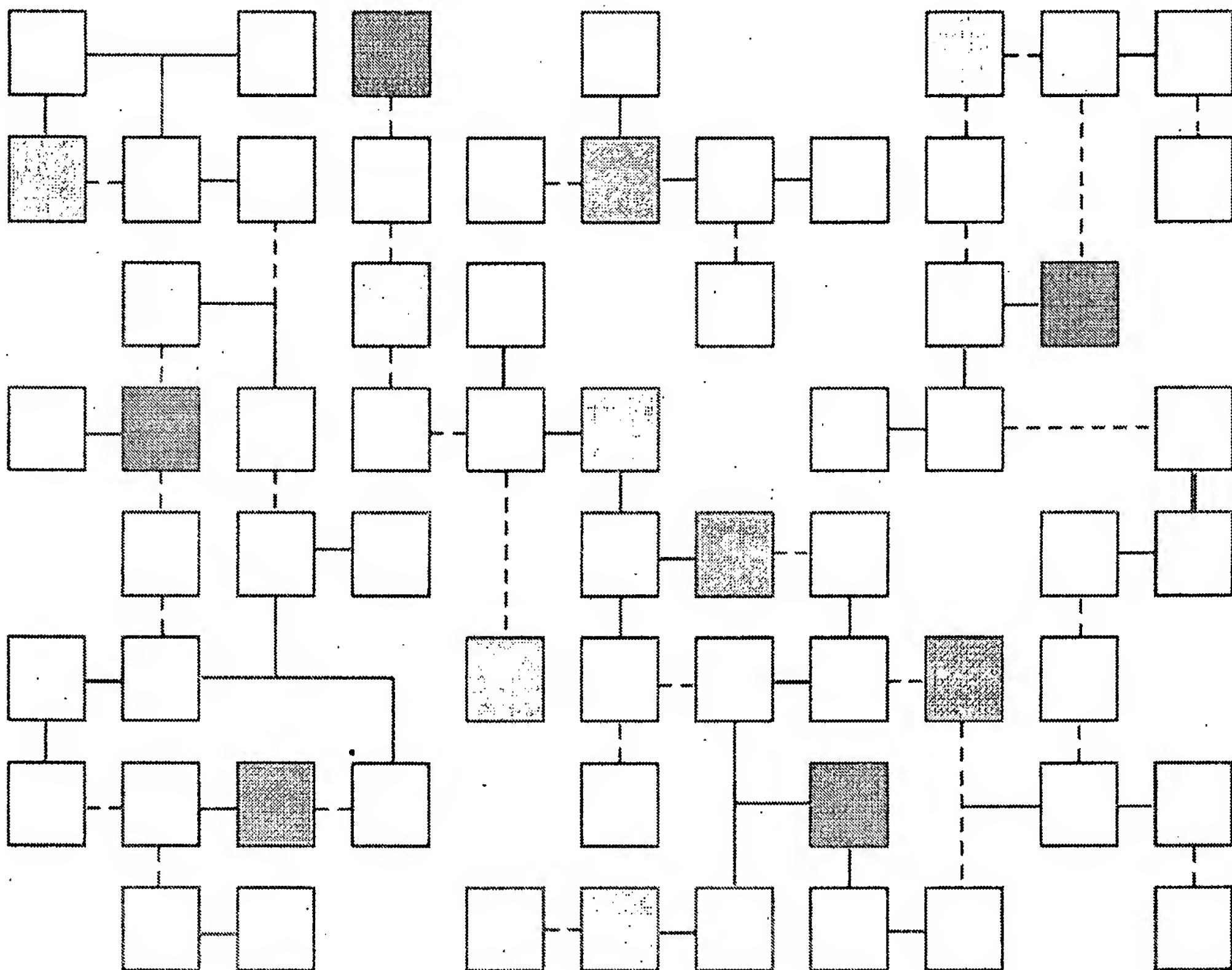
For a given user, information for a particular session used to track sign-ons.

Column Name	Title	Format	Text
AREA_ID		NUMBER	The identifier of the area to which this user session belongs.
EXPIRE_DATE		DATE	The date and time when this user session expires.
PROJECT_ID		NUMBER	The identifier of the project to which this user session belongs.
SECURITY_USER_ID		VARCHAR2(30)	The identifier of the security user to whom this user session belongs.
SESSION_ID		VARCHAR2(30)	A unique internal key for this user session.
USER_ROLE		VARCHAR2(18)	The role that the user has chosen for this session.

REF: US PATENT APPN.
SER NO.
10/767,511

EXHIBIT C
92 PAGES

ExtraView™ Administrator's Guide Version 3.1.2.1





Sesame Technology
269 Mount Hermon Road, Suite 205
Scotts Valley, CA 95066

Telephone: (831) 461-7100
Fax: (831) 461-7104
E-mail: info@sesame.com
www.sesame.com
© 1999 - 2001 Sesame Technology
All rights reserved

Manual Name: ExtraView Administrator's Guide
Revision Date: July 24, 2001

Information contained in this document, and the software to which it refers is subject to change without notice. This includes URL and any other web sites that may be mentioned. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise) or for any purpose, without the express written permission of Sesame Technology.

Sesame Technology may have patents, patent applications, trademarks, trademarks applied for, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Sesame Technology, the furnishing of this document does not give you any license to these patents, trademarks, copyrights or other intellectual property.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

About This Guide

The ExtraView Administrator's Guide gives users of the ExtraView Web-based issue tracking and workflow management system the knowledge and proficiency needed to accomplish two general goals:

1. Customize ExtraView to conform to their company's workflow processes, business rules, and inter-organizational vocabulary, with all the appropriate data fields and security privileges that are required.
2. Give Administrators the ability to successfully administer ExtraView on an ongoing basis in order to efficiently respond to user inquiries, concerns, and requests.

In writing this guide, we anticipate that the reader is at least somewhat familiar with many of the standard issue tracking functions found in ExtraView. Accordingly, this guide will assume this familiarity, and will focus on describing the Administrative functions of ExtraView.

Separate guides exist for the installation of ExtraView on your platform (not required if you are using Sesame's hosting service), for end users of ExtraView, and for users of the Application Programming Interface.

Guide Organization

The information in the ExtraView Administration guide is organized in the following manner:

Installation & Setup	This section will give Administrators all of the information needed to begin customizing their ExtraView installation.
Standard Configuration	Provides information and instructions on how to alter the standard ExtraView fields to suit your terminology.
Users & Security	Provides information on adding users to the system, adding user groups to the system, and granting permissions to users based on user groups.
Business Rules & Workflow	Describes how to use your existing workflow in conjunction with ExtraView.
Email Functionality	Gives detailed information on all of Extra View's Email notification features.

Notational Conventions

This guide follows certain notational conventions that are explained below:

- Terminology that Administrators can customize in ExtraView will be marked in *italics*:

Select a *Product* from the list.

- Names of buttons, links, lists, or fields will appear in **bold**:

Select a value from the **Owner** dropdown list.

- Whenever there are multiple steps involved in achieving a certain result, these will be noted numerically:

1. Click the **Edit** button.
2. Select a value from the list.
3. Click the **Update** button.

Table of Contents

About This Guide	iii
Guide Organization	iii
Notational Conventions	iv
Table of Contents	v
INTRODUCTION	2
ExtraView Product Description	2
Key Concept for Understanding ExtraView	2
ExtraView Installation and Setup	2
Defining a Process That Works for Your Company	3
Data Dictionary	3
Flexibility of the User Interface	4
User Groups and the Security System	4
Screen & Report Layout Editor	4
Interest Lists	5
Reporting	5
Application Programming Interface (API)	5
INSTALLATION & SETUP	6
System Customization	6
Installation Setup	6
Data Dictionary	9
Edit Data Dictionary Items	9
User Defined Fields (UDFs)	13
Editing an Existing User-Defined Field	16
ExtraView Layout Editor	17
Adding Layouts	17
Editing Screen Layouts	19
Creating Release and Module Layouts	23
To Add the Release Layout to Appropriate Screens:	24
Editing Fields	24
Deleting Fields	25
Clearing Layouts	25
Deleting Layouts	26
Allowed Value Types	26
User and System Wide Reports	32
Sign On Message	34
Behavior Settings	35
Default Values & Behavior Settings (Application Defaults)	35
Viewing Usage Statistics	39
ExtraView Configuration	40
Altering Configuration Settings	41
User Roles	45
Customized ExtraView Home Screen	46
Editing Your Personal Options	49
MANAGE USERS & SECURITY	51
User Groups	51

Add User Groups	51
Add New Users	52
Edit User Accounts	56
Delete User Accounts	57
System Security Keys	58
Edit an Existing Security Key	58
Add a New Security Key	58
Grant Security Permissions	59
Edit Security Privileges	59
Make a Field or Option Read Only	61
Make a Field or Option Write Only	61
Make a Field or Option Read and Write for a Particular User Group	62
Make a Field Invisible to a Particular User Group	62
Company Name Security	62
Manage Connected Users	63
Disconnect Inactive Users	64
Privacy Groups	64
Add a Privacy Group	65
BUSINESS RULES & WORKFLOW	67
Creating State Change Rules	67
Create State Change Rules	67
Customize State Change Rules	68
Customize Business Rules	69
Issuing Batch Commands	70
Issue Batch Commands	70
EMAIL FEATURES	73
Administrator-Defined Features	73
User-Defined Features	73
Turning System-Wide Email On or Off	73
Disable Automatic Email Generation for User Groups	74
Disable Include Guests for Screens or User Groups	75
Assign Module Owners	76
Set Product Email Address	76
Customize Email Subject Line	76
Notify of Own Updates	77
Select Email Format	78
Disable Automatic Email Generation	80
Disable Generate Email to External Users	80
Enable CC Email	81
Email Interest Lists	81
Create Normal Configuration Item Interest Lists	81
Create User-Defined Field Interest Lists	83
Edit Personal Email Interest Lists	85
Remove User ID from Interest List	85
ExtraView Help	86

INTRODUCTION

ExtraView Product Description

ExtraView is a Web-based issue tracking system that is designed to meet the following objectives:

- Easy to install, configure and administer, minimizing your organization's setup and ongoing cost of ownership
- Provide functionality that is easily extensible over time
- Able to support your processes and your workflow, without major modification
- Scalable to support large numbers of users and issues
- Easily customized to reflect your company's terminology, and data hierarchies, and able to provide extensive validation for data that describes your organization, products, and services

Key Concept for Understanding ExtraView

ExtraView Installation and Setup

This is a process that is best executed with some advance planning. The purpose of this guide is to give you complete details on the administrative portion of the initial setup and as well as ongoing support for your installation. The basic workflow suggested for setting up your installation is as follows:

- Plan your server hardware and network connectivity to support ExtraView. Sesame's technical support personnel can help with recommendations for suitable platforms
- Install Oracle and the Web server software on this hardware, and establish communications on your company's network or over the Internet
- Install the ExtraView application within Oracle. Please refer to the appropriate ExtraView installation guide for your platform
- Set up ExtraView. This Administrator's Guide covers the design of your system and how to provide the features you need for your company. Briefly, these are:
 - Defining and implementing the various groups or categories of users who will access the system

- Defining and creating the fields in your system
- Defining the relationships between fields
- Designing and laying out screens to support the fields you create
- Creating a structure of permissions that support access to each screen for each user group that was defined
- Setting up the workflow you want to use to control the processes in your company
- Designing standard reports for your users
- Adding user accounts to the system
- Testing the completed system

Defining a Process That Works for Your Company

ExtraView allows the System Administrator to define a process that conforms to the way the company works. It does not impose a fixed methodology on the company. The administrator can, without programming, set up rules appropriate to the company's needs.

Each issue you submit can be moved between any number of states, with each state being visible only to the group of users that are working on an individual state. For example, an *open* problem may only be viewed by the engineering group, who may only mark it *fixed* or *problem not found* after working on it. They could not, for example, *close* the problem since that is a state defined to a different user group.

A user group is created for all people who should follow the same rules. Typically these would fall along the lines of customers, support staff, engineering, quality assurance, etc., but complete flexibility exists to define what user groups you create and how many you create.

Should you have a workflow process that cannot be accommodated within ExtraView's standard functionality, the product can be simply extended with additional source code. ExtraView was designed to make it easy to alter or add functionality within the source code.

Data Dictionary

The data dictionary is the central place where all field definitions are stored and maintained. In addition, this core component of ExtraView controls many of the attributes of each field, such as its display type, display title, whether the field is selectable on reports, the SQL used to populate the field if it's a list, default value, and help text.

Flexibility of the User Interface

ExtraView can be modified in a number of ways in order to tailor its look and feel to any company's needs.

The following changes can be made simply, either by you or by Sesame Technology:

- Alter screen colors and fonts
- Place the system menu horizontally or vertically on the screen
- Add your company logo
- Edit all text labels to reflect your own terminology
- Rename, add, or delete menu items
- Create new menu buttons in any style

User Groups and the Security System

This inter-related group of concepts is central for understanding ExtraView. Individual users belong to one or more user groups and share the same characteristics and permissions. For example, one user group may have read and write access to a particular field, while another group may only be able to view the same field, while still another group may not even be able to see the same field.

Any field that is available in ExtraView has a security key to protect it. They can either be whole menus, or individual fields within a screen. For example, a security key exists for accessing the security module itself. Another security key exists for the product menu item on the Administration menu. Another example is that a security key exists to control access to the description field. Basically, all of the fields that are visible on ExtraView can be turned on or off based on user group privileges.

The Grant Security Privileges section controls all these accessibility features in your version of ExtraView. In a matrix view, the intersection of the security key and the user group is a read and write switch. Therefore, for every item that has a defined security key, you can allow or prohibit any user group to access that feature.

Screen & Report Layout Editor

This component of the administration allows you to set up and alter the layout of the *Add Problem* and *Edit Problem* screens. In addition, key reports such as the Quick List and the Detailed Report also use layouts defined by this function. Different layouts can be defined for different user groups within your system, offering a tremendous amount of flexibility. Each layout works in conjunction with security permissions for each field. Therefore, an important concept to understand is that simply placing a field on a screen does not automatically give all users the ability to read or write to the field. Using the

Grant Security Privileges option allows you to define which fields are visible and updateable to each group of users. The security privilege for the field overrides the fact that a field may be placed on a screen or report.

Interest Lists

These are a powerful feature of ExtraView, ensuring automatic notification of events to appropriate individuals within an organization. Interest lists can be defined within the data dictionary on any field that may have a list of values. For example, you may define an interest list that notifies individuals on all issues that have a severity level of *critical*, or you may define an interest list that notifies individuals whenever an issue affects a particular *module*.

Reporting

Within the ExtraView administration, Reports can be defined to be available to the entire community of users. Reports can be simply a set of filters that are used to query the ExtraView database, or they can be coupled with internal or customized layouts.

Application Programming Interface (API)

The API to ExtraView allows the user to extend ExtraView's functionality. The key features of the API are:

- A full Command Line Interface (CLI) that allows users to access functions such as adding, updating, deleting and searching from a command line. This is typically used from a UNIX, Linux or Windows NT command shell.
- A set of URL functions which access ExtraView to perform functions. This eliminates the need for expensive software such as SQLNet on each client computer.
- An extendible interface allowing the user to create additional functions with source code. This can be used, for example, to integrate other applications, such as CRM and SCM directly into ExtraView.

INSTALLATION & SETUP

The ExtraView Installation and Setup features are designed to provide the System Administrator with a toolset for making even the most highly custom design and configuration requirements easy to implement, with little or no programming required.

These same features also make it possible for the Administrator to continue to customize and refine the system to better conform to the company's changing needs, with little or no downtime.

Note: Because initial installation and setup requirements will normally have been implemented by the Sesame Technical Support team at the time of license, it may be advisable for the System Administrator interested in first learning day-to-day system management functions to skip this chapter for now, and begin instead with chapter 2, Users and Security.

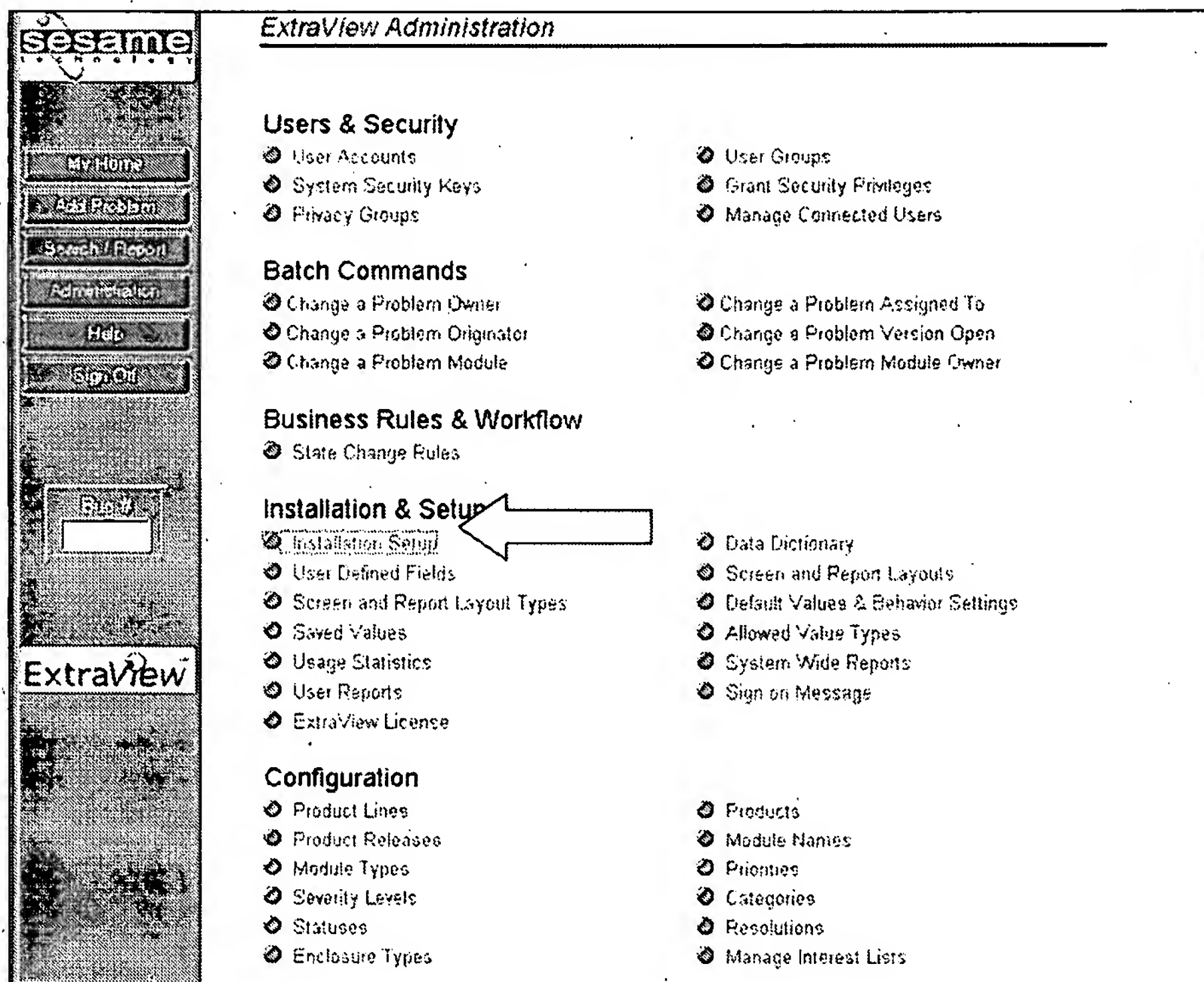
System Customization

The following topics will guide you through your ExtraView customization process: Data Dictionary, User-Defined Fields, Layout Editor, Allowed Value Types, Sign On Message, Behavior Settings, Usage Statistics, Configuration Settings, User Roles, Home Screen, and Editing Personal Options.

Installation Setup


















This section allows you to set up and modify system level information that is used throughout the ExtraView system. This is one of the key places where installation information is altered. A Sesame Technology representative will enter most of this information for greater customer convenience.

1. Installation Setup is found under the Installation & Setup sub-heading on the Administration menu.



ExtraView Administration menu

The following screen appears:

<i>Installation Setup - Installation</i>			
	<i>Database field name</i>	<i>Value</i>	<i>Description</i>
	COMPANY_NAME	Redback	
	COMPANY_ADDRESS1		
	COMPANY_ADDRESS2		
	COMPANY_CITY		
	COMPANY_STATE	CA	
	COMPANY_ZIP		
	COMPANY_PHONE	(831) 461-7100	
	COMPANY_EMAIL		
	WINDOW_BG_COLOR	white	Window background color
	BG_COLOR	#EEEEEE	Background color for tables
	BG_ALT_COLOR	#D9E0E4	Alternative background color for tables
	ALT_COLOR	white	Alternate color for add, edit and search screens
	LABEL_COLOR	royalblue	Color of field labels on the screen
	TITLE_COLOR	scarlet	
	DEFAULT_FONT	Arial, Helvetica, sans-serif	
	MENU_DIRECTION	VERTICAL	The main menu can be HORIZONTAL or VERTICAL
	SESSION_EXPIRE_TIME_HOURS	24	Max session idle time before user is forced to re-login

Installation Setup screen

- Clicking the **Edit** icon next to an item allows you to make changes to the default settings. For example, this is the screen for editing WINDOW_BG_COLOR, which is the Window Background Color.

<i>Change entry in the installation details table</i>			
Fixed database name	WINDOW_BG_COLOR		
Description	<input type="text" value="Window background color"/>		
Value	<input type="text" value="white"/>		
<input type="button" value="Update"/> <input type="button" value="Cancel"/>			

Editing Installation Details

- To edit this value or any of the others in this section just change the Description and or Value and click the **Update** button.
- Below is a complete list of Installation Setup Database Field Names, sample values and their appropriate descriptions; they can be altered in the Installation & Setup option.

<i>Application Default Name</i>	<i>Typical Value</i>	<i>Description</i>
COMPANY_NAME	Your Company Name	Company Name
COMPANY_ADDRESS1	269 Mt Hermon Rd	Address 1

COMPANY_ADDRESS2		Address 2
COMPANY_CITY	Scotts Valley	Company City
COMPANY_STATE	CA	State
COMPANY_ZIP	95066	Zip Code
COMPANY_PHONE	(831) 461-7100	Phone Number
COMPANY_EMAIL	support@sesame.com	Email Address
WINDOW_BG_COLOR	White	Window background color
BG_COLOR	Gainsboro	Background color for tables
BG_ALT_COLOR	Lightsteelblue	Alternative background color for tables
ALT_COLOR	White	Alternate color for add, edit and search screens
LABEL_COLOR	#0000FF	Color of field labels on the screen
TITLE_COLOR	Scarlet	Color for titles on the screen
DEFAULT_FONT	Arial, Helvetica, Sans-serif	Default font for text in the system
MENU_DIRECTION	VERTICAL	The main menu can be HORIZONTAL or VERTICAL
SESSION_EXPIRE_TIME_HOURS	24	Max session idle time before user is forced to re-login.
SITE_URL	http://www.extraview.net	The URL for your ExtraView site.

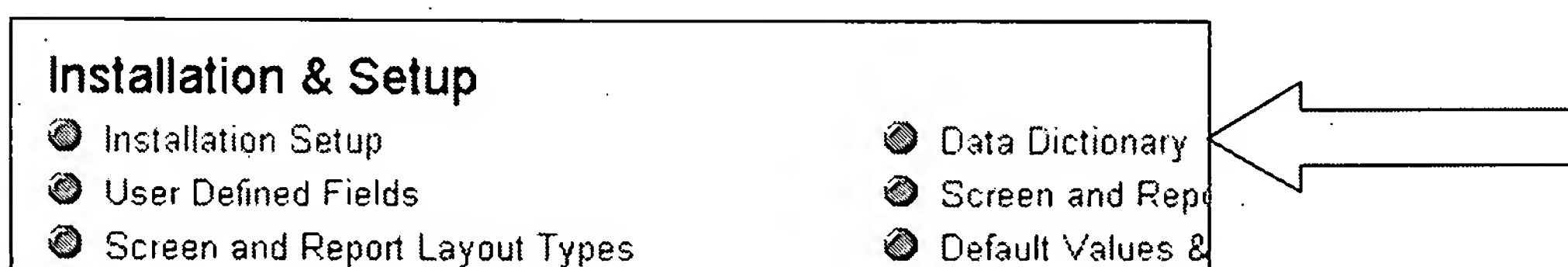
Data Dictionary

The data dictionary is the central place where all field definitions are stored and maintained. In addition, this core component of ExtraView controls many of the attributes of each field, such as:

- display type.
- display title.
- whether the field is selectable on reports.
- the SQL used to populate the field if it's a list.
- default values.
- help text and more.

Edit Data Dictionary Items

1. From the Administration menu, click on **Data Dictionary**.



The following screen appears:

Data Dictionary						
Add Add a new field to the database						
<i>Fixed name</i>	<i>Screen title</i>	<i>Type</i>	<i>Help Tip</i>	<i>Created</i>	<i>Last updated</i>	
Edit ACTIVE	Active	Database	Whether the item is active	system 28-MAY-2000 11:20	system 21-DEC-2000 22:34	
Edit ALT_ID	Alt ID	Database	This is the alternative ID for a problem	system 28-JUN-2000 14:13	system 07-MAR-2001 14:55	
Edit AREA	Area	Database		system 28-MAY-2000 11:20	system 28-JUN-2000 15:10	
Edit ASSIGNED_TO	Assigned To	Database	The name of the user that this problem is currently assigned to.	system 28-MAY-2000 11:20	fio 20-JUN-2001 11:20	
Edit ASSIGNED_TO_NAME	Owner	Database		system 30-NOV-2000 10:50	fio 12-JUN-2001 08:46	
Edit ATTACHMENT	Attachments	Database	The file or files that you are able to attach to a problem.	system 08-JUL-2000 18:57	system 10-DEC-2000 11:55	

Data Dictionary Summary Screen

- Click the **Edit** button next to the item that you want to modify and then press **Update** to save the changes.
- You can include or alter many different features from this screen; see the screenshots below, with their corresponding roman numerals.

Change a database dictionary entry

Fixed database name: ASSIGNED_TO

Title to display: **I**

Data Type: ☒ DATABASE ☐ UDF ☐ LABEL ☐ VALUE **II**

Required field: ☐ Yes ☒ No **III**

Display Type: ☒ TEXTFIELD ☐ TEXT ☐ LOGAREA ☐ PRINTTEXT ☐ LABEL ☐ LIST ☐ TAB **IV**

Display as URL: ☐ Yes ☒ No **V**

URL: **VII**

Allow selection on reports: ☐ Yes ☒ No **VI**

Save last value: ☐ Yes ☒ No

Allow new entries to be added dynamically to list: ☐ Yes ☒ No **VIII**

Enable interest list on this field: ☐ Yes ☒ No

Default size (characters):

Default format:

SQL: **III** **IV**

Order by: **V**

First Parent Field Name:

First Parent SQL:

Second Parent Field Name:

Second Parent SQL:

Help Text: **XI**

Help URL: **XII**

X

Edit Data Dictionary Item Screen

- I. The Screen Title for any field in your ExtraView system. Just type a title in the given field and update. The field name will instantly be changed throughout ExtraView.
- II. The Data Type. Once set, these should not be altered. Changing these may have disastrous consequences on the use of ExtraView.
 - a. DATABASE. These are ExtraView's inbuilt database fields. Their type must never be changed.
 - b. UDF. These are the User Defined Fields in your system.
 - c. SCREEN. These are the names of the screens within the ExtraView system.
 - d. LABEL. These are fields created purely as informational labels on screens, and for which no value is stored.
- III. Required Field. You have the ability to set a field as required or not.
- IV. The Display Type. This can be changed into any one of the following:

- a. TEXTFIELD. This is a normal, one line text field. Data may be up to 255 characters in a TEXTFIELD.
- b. TEXTAREA. This is a larger, multi-line text field; it is expandable and collapsible and can hold up to 32k of text.
- c. LOGAREA. Similar to a TEXTAREA but previous entries to the field cannot be edited. You will see the User ID of the person who updated the field and the time and date when it was updated. It thus functions as a log of successive entries to the field.
- d. PRINTTEXT. This is also similar to the TEXTAREA field, but on display, a fixed-width font is used for the field. This can be used if the field may routinely be used to hold diagrams drawn with characters such as +-----+ on the keyboard and you want to preserve the accuracy of the diagram.
- e. LABEL. A simple form label.
- f. LIST. A list of values; you can add or delete list items from the Configuration screen or from the User Defined Fields section.
- g. TAB. A list of values that is displayed as a set of tabs across the screen. Typically this is used to provide a high-level selection on a screen, where the subsequent fields depend on the tab selected. You should not use this display type if the supporting list has more than six entries.
- h. NUMBER. A field that accepts and stores only numbers.
- i. DATE. This is a field that allows dates to be entered and stored.

V. The ability for a field to be associated with a URL.

- a. From the Data Dictionary click the Display as URL radio button to YES

☐ NUMBER ☐ DATE
 Display as URL ☒ Yes ☐ No
 URL

- b. Type the appropriate URL in the URL field below the Display as URL button
- c. The URL can be anything you like, as long as it ends with **?p=\$\$VALUE\$\$**. This will take the value that is in the field and attach it to the appropriate URL.

Examples:

- i. [http://search.yahoo.com/search?p=\\$\\$VALUE\\$\\$](http://search.yahoo.com/search?p=$$VALUE$$) - After pressing the URL button Yahoo search results will be returned for the value that is in the particular field.

- ii. **\$\$OAS\$\$se_security_user.showUserDetails?p_user_id=\$\$VALUE\$\$** - This example will display a particular user's details. By pressing the URL button a screen will be returned showing information on the user, from the ExtraView Database.
- VI. **Allow Selection on Reports.** This will enable the particular field to be available on Additional Reports.
- VII. **Save Last Value.** This will allow the user to save the previous value in the list when adding, editing or deleting.
- VIII. **Enable Interest List.** By choosing YES you are able to go into the configuration for this particular field and create an Interest List based on a particular value.
- IX. **Default Value.** From here you have the ability to enter a default value for a particular field. This value will have to be the actual database name as opposed to the title and will be automatically selected each time you add a problem to the database.
- X. **SQL Statements.** This shows the SQL statement used to populate the particular list. These fields also give you the option of inputting Parent Field Names and Parent SQL as well as Child Field Names and Child SQL in order to create dependencies.
- XI. **Help Text (tool tip).** When you mouse over a field, this is the message that will appear.
- XII. **Help URL.** You can link this to a field or page in your online help system. If you are an ExtraView hosted customer, note that this URL need not be on Sesame's server. You can store and access these files stored anywhere and accessible over the Internet.

User Defined Fields (UDFs)

A User Defined Field is a field that is setup specifically for your ExtraView installation and does not exist in the basic product. Since users may want to further customize their own site to include fields that are more specific and appropriate to their needs, **UDFs** are available to satisfy this requirement. This is a highly efficient and extensible mechanism. There is no limit to the number of UDFs that may be created.

1. User Defined Fields are found under the Installation & Setup sub-heading on the Administration menu.

Installation & Setup

- Installation Setup
- **User Defined Fields**
- Screen and Report Layout Types
- Saved Values
- Usage Statistics
- User Reports
- ExtraView License
- Data Dictionary
- Screen and Report Layouts
- Default Values & Behavior Settings
- Allowed Value Types
- System Wide Reports
- Sign on Message

Installation & Setup section

2. Click on **User Defined Fields**. The following screen appears:

User Defined Fields

Add Add a new entry to the User Defined Fields table

	Fixed Name	Screen Title	Data Type	Help Text	Help URL	Created
EDIT	COMMENTS	Comments	LOGAREA			system 21-DEC-2000 10:24
EDIT LIST	COMPONENT	Component	LIST			system 17-JUL-2000 19:12
EDIT LIST	CUSTOMER	Customer	LIST			system 21-DEC-2000 10:26
EDIT	TEST_CASE_ID	Date/IDF	DATE	If you enter a Test Case ID, you must enter a Test Case Location		system 05-JAN-2001 23:45
EDIT	DESCRIPTION	Description	TEXTAREA			system 21-DEC-2000 10:23
EDIT LIST	OS	OS	LIST			system 21-JUL-2000 23:07
EDIT LIST	PLATFORM	Platforms	LIST			system 08-JUL-2000 16:59
EDIT	RELEASE_NOTES	Release Notes	TEXTAREA			system 21-DEC-2000 10:25
EDIT	CLARIFY_ID	Serial #	TEXTFIELD			system 07-SEP-2000 16:48
EDIT	TEST_CASE_LOCATION	Test Case Location	TEXTFIELD	If you enter a Test Case Location, you must enter a Test Case ID		system 05-JAN-2001 00:00
EDIT	WORKAROUND	Workaround	TEXTAREA			system 21-DEC-2000 10:24

11 records selected

Return

Add User-Defined Fields screen

To add a new **UDF**, click on the **Add** icon:

Add entry to the User Defined Fields table

Name

Title

Data table name

Column name in table

Data Type ☒ TEXTFIELD ☐ TEXTAREA ☐ LOGAREA ☐ NUMBER ☐ DATE ☐ LIST ☐ TAB ☐ URL

Required field ☐ Yes ☒ No

Display as URL ☐ Yes ☒ No

URL

Allow selection on reports ☐ Yes ☒ No

Allow new entries to be added dynamically to list ☐ Yes ☒ No

Help text

Help URL

Add a New UDF

3. Enter a Name (which will become the database name), Title (which will appear on the screens), and select a Data Type. The remaining items are optional, however you may want set Allow Selection on Reports to "Yes" to ensure the new UDF will appear in the various ExtraView reports. Any item that appears in bold on the screen is not optional, and is needed to continue any further within the site. Click **Update** when finished.

It is very important to note that UDF's can be created in the manner described above, but are not yet available to users on their screens. To achieve this, two further steps must be taken. After creating the UDF you must place the field on a layout at the appropriate place and you must then make the fields visible to the appropriate user groups, via the Grant Security Privileges screen.

Name	<input type="text" value="complete_date"/>
Title	<input type="text" value="Project Completion Date"/>
Data table name	<input type="text"/>
Column name in table	<input type="text"/>
Data Type	<input type="radio"/> TEXTFIELD <input type="radio"/> TEXTAREA <input type="radio"/> LOGAREA <input type="radio"/> NUMBER <input checked="" type="radio"/> DATE <input type="radio"/> LIST <input type="radio"/> TAB <input type="radio"/> URL
Required field	<input type="radio"/> Yes <input checked="" type="radio"/> No
Display as URL	<input type="radio"/> Yes <input checked="" type="radio"/> No
URL	<input type="text"/>
Allow selection on reports	<input checked="" type="radio"/> Yes <input type="radio"/> No
Allow new entries to be added dynamically to list	<input type="radio"/> Yes <input checked="" type="radio"/> No
Help text	<input type="text"/>
Help URL	<input type="text"/>
<input type="button" value="Update"/> <input type="button" value="Cancel"/>	

Creating a New UDF

Editing an Existing User-Defined Field

To edit an existing UDF:

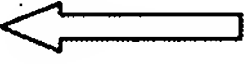
1. Click the **Edit** icon next to the UDF you wish to change, which brings up the following screen:

Change an entry in the User Defined Fields table	
UDF Name	CUSTOMER
Screen Title	<input type="text" value="Customer"/>
Data Table Name	<input type="text" value="PROBLEM_UDF"/>
Column Name in Table	<input type="text"/>
Data Type	LIST
Required field	<input type="radio"/> Yes <input checked="" type="radio"/> No
Display as URL	<input type="radio"/> Yes <input checked="" type="radio"/> No
URL	<input type="text"/>
Allow selection on reports	<input type="radio"/> Yes <input checked="" type="radio"/> No
Allow new entries to be added dynamically to list	<input type="radio"/> Yes <input checked="" type="radio"/> No
Help text	<input type="text"/>
Help URL	<input type="text"/>
<input type="button" value="Update"/> <input type="button" value="Delete"/> <input type="button" value="Cancel"/>	

Edit an existing UDF

2. Make the desired changes, and then click the **Update** button.

Note: You cannot change the UDF data type from this screen. This can only be done in the Data Dictionary. See Screen shot below.

UDF Name	CUSTOMER
Screen Title	Customer
Data Table Name	PROBLEM_UDF
Column Name in Table	
Data Type	LIST 
Required field	<input type="radio"/> Yes <input checked="" type="radio"/> No
Display as URL	<input type="radio"/> Yes <input checked="" type="radio"/> No
URL	
Allow selection on reports	<input checked="" type="radio"/> Yes <input type="radio"/> No

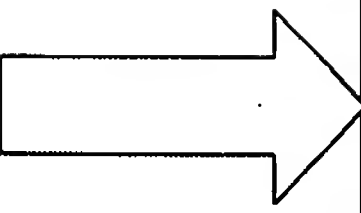
Edit UDF Screen

ExtraView Layout Editor

The **Layout Editor** allows you to create your own customized screens in ExtraView. You are able to add fields for specific user groups or you can add fields on a default basis and control user access through Grant Security Privileges.

Adding Layouts

1. The first step is to define the screen layout types you will be using. These are limited to the Add, Edit, Release and Search screens, however all ExtraView reports will soon be integrated into the Layout Editor. From the ExtraView Administration screen, select Screen and Report Layout Types, in the Installation & Setup section.



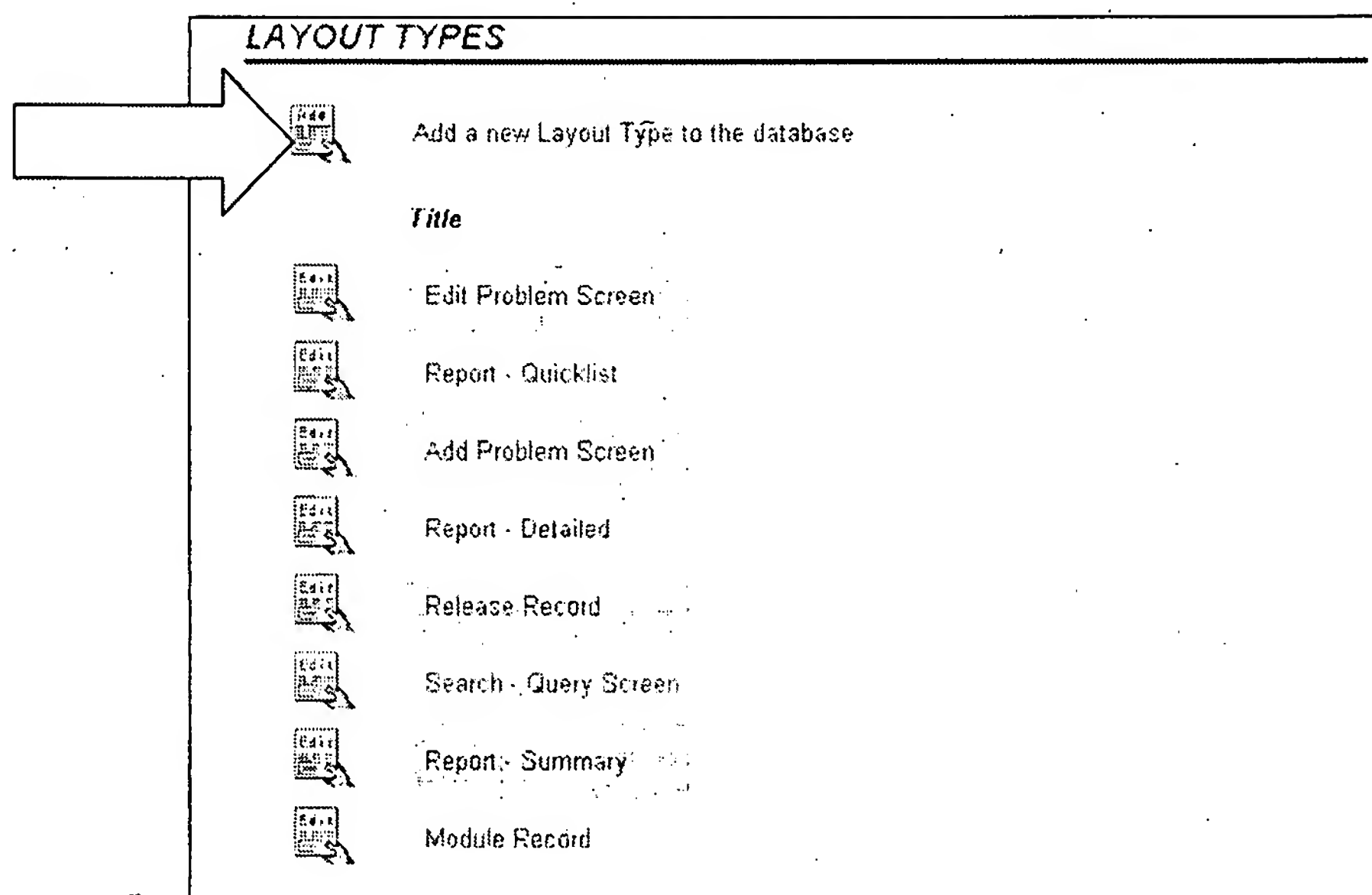
Installation & Setup

- ☐ Installation Setup
- ☐ User Defined Fields
- ☒ Screen and Report Layout Types
- ☐ Saved Values
- ☐ Usage Statistics
- ☐ User Reports
- ☐ ExtraView License

- ☐ Data Dictionary
- ☐ Screen and Report Layout Editor
- ☐ Default Values & Behavior Settings
- ☐ Allowed Value Types
- ☐ System Wide Reports
- ☐ Sign on Message

ExtraView Administration Screen

2. Click on the **Add** icon to add a new layout type.



Screen and Report Layout Types

3. Enter the fixed database name and an appropriate screen title to display. The fixed database names are: ADD_PROBLEM, EDIT_PROBLEM, RELEASE, SEARCH, QUICKLIST, DETAILED_REPORT, SUMMARY_REPORT, and MODULE.
4. Click the **Update** button when finished.

The screenshot shows a dialog box titled "Add a new Layout Type to the database". It contains two text input fields: "Fixed Database Name" and "Title to Display". Below these fields are two buttons: "Update" and "Cancel".

Adding a new Layout

Editing an existing layout type is accomplished in a similar manner.

1. Click the **Edit** icon next to the layout type you wish to change.
2. You can now delete the layout by pressing the **Delete** button, or change the screen title to display. You cannot change the fixed database name.

3. When you are finished, press the **Update** button.

Change Layout Type details

Fixed Database Name ADD_PROBLEM

Title to Display

Editing an existing Layout

Editing Screen Layouts

Once you have created the layout, return to the ExtraView Administration screen, and select **Screen and Report Layout Editor** from the Installation & Setup section.

Installation & Setup

☒ Installation Setup

☒ User Defined Fields

☒ Screen and Report Layout Types

☒ Saved Values

☒ Usage Statistics

☒ User Reports

☒ ExtraView License

☒ Data Dictionary

☒ **Screen and Report Layout Editor**

☒ Default Values & Behavior Settings

☒ Allowed Value Types

☒ System Wide Reports

☒ Sign on Message

ExtraView Administration screen

1. If you would like to create ExtraView screens for specific user groups, then you can select the Security Group to which the layout will apply from the dropdown list at the top. Additionally, you can use the default layout and control User Group screen and field access via the Grant Security Privileges section of the ExtraView Administration screen.

Layouts

Select the security user group to which the layout belongs. If no layout is specified for a user group, the default layout will apply.

Select Security Group * Default Layout for All Security Groups *

Now choose a new layout to add to the above security group, or press the edit button by an existing layout to alter that layout.

Add a new layout for the entire system

QA	
Customer	
Manager	
Test	
Guest	
Administrator	
Engineer	

Add Problem Screen Add Screen

Edit Problem Screen Edit Screen

Search - Query Screen Search Screen

0 records selected

Selecting a Security Group

2. Select the layout type to customize. The names in this field are the screen titles you chose when creating the layouts.

Layouts

Select the security user group to which the layout belongs. If no layout is specified for a user group, the default layout will apply.

Select Security Group * Default Layout for All Security Groups *

Now choose a new layout to add to the above security group, or press the edit button by an existing layout to alter that layout.

Add a new layout for the entire system

Layout Type	Description
* Select Layout Type to Add *	
* Select Layout Type to Add *	
Add Problem Screen	
Edit Problem Screen	
Module Record	
Release Record	
Report - Detailed	
Report - Quicklist	
Report - Summary	
Search - Query Screen	

0 records selected

Selecting a layout

For this example, we have selected *Add Problem*, which gives the following screen:

Note: You will have to repeat steps 4 thru 8 for the Edit Screen and the Search Screen.

Screen and Report Layout Editor

Layout Name: Layout Type: ADD_PROBLEM User Group: Default

Row Column Field Name:

Rowspan Colspan

Required Yes ☐ No ☒

Add Problem Screen Layout

The screen layout is based on a grid, so you will have to choose the appropriate columns and rows to designate where you want a particular field to be placed. In this example, we want to place the *Title* field (SHORT_DESCR is the database name) in the top left hand corner of the Add Problem screen.

- To do this, select the field name from the Field Name list (which displays all the fields available, sorted alphabetically by screen title). Ensure Row is set to 1, and Column is set to 1, this indicates that the field will appear in the first row and first column of the Add Problem form.

The Rowspan option will allow a field to span several rows.

The Colspan option will allow a field to span several columns.

In this case we want the *Title* field to be a long field, so we have set the Colspan to 2.

Screen and Report Layout Editor

Layout Name: Layout Type: ADD_PROBLEM User Group: Default

Row Column Field Name:

Rowspan Colspan

Required Yes ☐ No ☒

Building the Add Screen

- Click the **Update Cell** button when finished.

6. After updating the cell, you will see a screen that shows the layout of the Add Screen so far.

Screen and Report Layout Editor

Layout Name: Layout Type: ADD_PROBLEM User Group: Guest

	<input type="button" value="Ins"/> <input type="button" value="Del"/>	<input type="button" value="Ins"/> <input type="button" value="Del"/>
	Col 1	Col 2
<input type="button" value="Ins"/> <input type="button" value="Del"/>	Row 1	SHORT_DESCR

Row: Column: Field Name:

Rowspan: Colspan:

Required: Yes ☐ No ☒

Add Screen Layout

7. Continue adding fields to the Add Screen layout in the same manner.
- Fields such as *Description* and *Comments* and any other “text area” fields are automatically set to span several rows, and can be dynamically increased or decreased on the actual screen that is created. This being the case, leave the Rowspan set to 1.
 - The fields that have been designated as “Required” appear in bold.

When you have finished adding fields and you have saved all of your changes click the **Return** button at the bottom of the screen to return to the main Layouts page.

You can design additional screen layouts for the Edit and Search screens by selecting **Edit** and **Search** from the dropdown list and following steps 4 to 8.

Layouts

Select the security user group to which the layout belongs. If no layout is specified for a user group, the default layout will apply.

Select Security Group

Now choose a new layout to add to the above security group, or press the edit button by an existing layout to alter that layout.

Add a new layout for the entire system

Layout Type	Description
* Select Layout Type to Add *	
* Select Layout Type to Add *	
Add Problem Screen	
Edit Problem Screen	
Module Record	
Release Record	
Report - Detailed	
Report - Quicklist	
Report - Summary	
Search - Query Screen	

ds selected

Adding Layouts screen

Creating Release and Module Layouts

The Release and Module layout are special cases. These are not individual screens, but a screen layout embedded within another screen layout. Layouts can be designed that can be embedded within both the Add and Edit screens if desired. You should be aware that you should not create recursion by embedding a layout within itself. ExtraView will only allow layouts to be nested four levels deep to prevent this behavior.

Since issues can be tracked with ExtraView at the release or module level as well as at the problem level, release and module data is grouped in a "release row" or "module row" below the main problem data and above the long text areas, such as Description. After the single-row release or module layout is created, these fields will appear on the Add and Edit screens according to how the permissions are set in Grant Security Privileges.) This is discussed in detail in the next section.

For this example, we will use the Release Layout.

If you are planning on having fields with Release Information just do the following:

1. Click on Layout Editor from the Administration menu.
2. Select **Release Record** from the dropdown list.

Layouts

Select the security user group to which the layout belongs. If no layout is specified for a user group, the default layout will apply.

Select Security Group

Now choose a new layout to add to the above security group, or press the edit button by an existing layout to alter that layout.

Add a new layout for the entire system

Layout Type	Description
* Select Layout Type to Add *	
* Select Layout Type to Add *	
Add Problem Screen	
Edit Problem Screen	
Module Record	
Release Record	
Report - Detailed	
Report - Quicklist	
Report - Summary	
Search - Query Screen	

us selected

Add Release Layouts

3. Add the fields that are appropriate to your install. These may include such things as:
 - a. RELEASE_FOUND
 - b. RELEASE_FIXED
 - c. RELEASE_ASSIGNED_TO
 - d. RELEASE_RESOLUTION
4. Save your current Layout by clicking the **Save Layout** button.

To Add the Release Layout to Appropriate Screens:

1. Click Administration and then Layout Editor.
2. For both the **Add** and **Edit** screen select LAYOUT.RELEASE from the dropdown list.
3. Choose the particular row that you want to add the release information to by selecting the row from the Row dropdown or by clicking the **Insert** button to insert a new row.
4. Save your changes and view your results.

Editing Fields

1. If you want to edit a particular cell, simply click on the field you want to edit.
2. You now have the ability to change the Rowspan, Colspan, Form Location or you can change the existing field with a completely different field.

3. Click the **Update Cell** button. Make sure you click **Save Layout** after each of your changes; otherwise changes will be lost when you leave this page.

Screen and Report Layout Editor

Layout Name: Layout Type: ADD_PROBLEM User Group: Guest

	<input type="button" value="Ins"/> <input type="button" value="Del"/>	<input type="button" value="Ins"/> <input type="button" value="Del"/>	<input type="button" value="Ins"/> <input type="button" value="Del"/>
	Col 1	Col 2	Col 3
<input type="button" value="Ins"/> <input type="button" value="Del"/> Row 1	SHORT_DESCR		
<input type="button" value="Ins"/> <input type="button" value="Del"/> Row 2	PRODUCT_NAME	CATEGORY	ORIGINATOR
<input type="button" value="Ins"/> <input type="button" value="Del"/> Row 3	MODULE_ID	PROBLEM_TYPE	
<input type="button" value="Ins"/> <input type="button" value="Del"/> Row 4	PRIORITY	SEVERITY_LEVEL	STATUS
<input type="button" value="Ins"/> <input type="button" value="Del"/> Row 5	PLATFORM	PRIVACY	
<input type="button" value="Ins"/> <input type="button" value="Del"/> Row 6	DESCRIPTION		
<input type="button" value="Ins"/> <input type="button" value="Del"/> Row 7	COMMENTS		

Row Column Field Name

Rowspan Colspan

Required Yes ☐ No ☒

Add Screen Layout

Deleting Fields

To delete a field:

1. Click on the particular field you want to delete.
2. Click the **Delete Cell** button.
3. Click **Save Layout**.

Clearing Layouts

To clear a layout:

1. Press the **Clear Layout** button
2. Press **Save Layout**

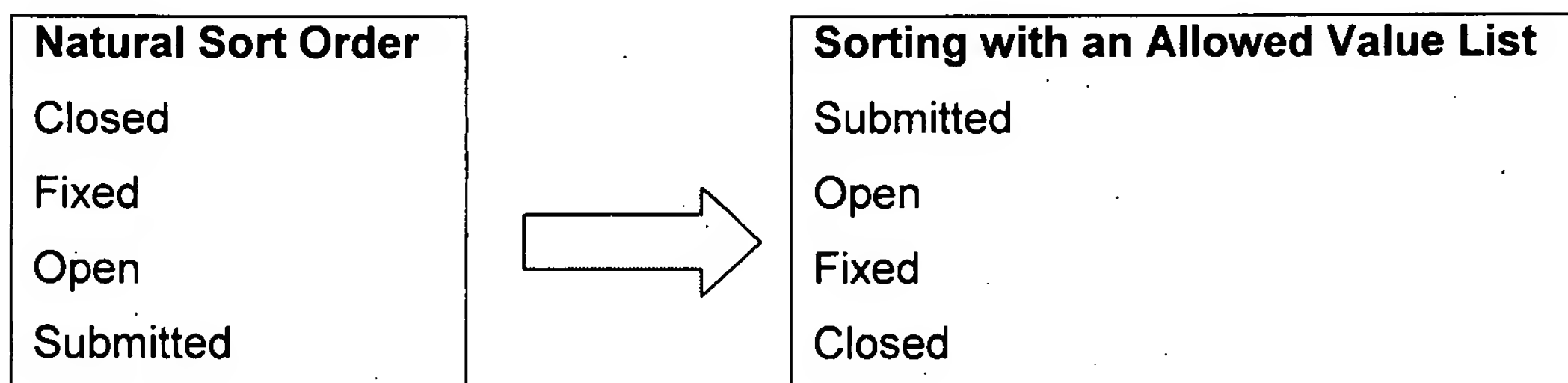
Deleting Layouts

To delete a layout:

1. Press the **Delete Layout** button.
2. Press **Save Layout**.

Allowed Value Types

Allowed Values serve two purposes for in your installation of ExtraView. It gives you the opportunity to sort your field lists in a non-natural sort way. For example, if you have a list of statuses as follows, they may be sorted as shown:



Allowed Value Lists also gives you the opportunity to have certain lists dependent on individual values in other lists; for example, a list of specific *Platforms* may only be displayed if the connected parent *Product* is first selected.

1. From the Administration menu, click **Allowed Values** under the Installation & Setup sub-section.

The following screen appears:

Add

Add a new allowed value type to the database

	Title	Parent DB Name	Child DB Name
<div>Edit</div> <div>List</div>	Category		Category
<div>Edit</div> <div>List</div>	Disposition		Disposition
<div>Edit</div> <div>List</div>	Product		Product
<div>Edit</div> <div>List</div>	Status		Status
<div>Edit</div> <div>List</div>	longallowedvaluestitlelong		Attachments
<div>Edit</div> <div>List</div>	module		Module
<div>Edit</div> <div>List</div>	parent		parent
<div>Edit</div> <div>List</div>	parent 2		parent2
<div>Edit</div> <div>List</div>	platform		Platforms
<div>Edit</div> <div>List</div>	severity		Severity
<div>Edit</div> <div>List</div>	Module Component	Module	A Component
<div>Edit</div> <div>List</div>	Product Platform	Product	Platforms
<div>Edit</div> <div>List</div>	module child	Module	child
<div>Edit</div> <div>List</div>	parent child1	parent	child
<div>Edit</div> <div>List</div>	parent child2	parent	child2
<div>Edit</div> <div>List</div>	parent2 parent	parent2	parent
<div>Edit</div> <div>List</div>	product parent2	Product	parent2

17 records selected

Return

Allowed Value Types screen

- To create a new Allowed Value Type, click the **Add** icon.

Allowed Value Types

Add

Add a new allowed value type to the database

Adding a New Allowed Value Type

The following screen appears:

Add a new Allowed Value Type to the database

Title	<input type="text"/>
Parent database name	* None *
Child database name	A Component(COMPONENT)
<input type="button" value="Update"/> <input type="button" value="Cancel"/>	

Add a new Allowed Value Type

3. The Title is whatever you wish to name your Allowed Value Type. If you wish to set up dependencies, select a Parent Database Name and a Child Database Name to establish an appropriate relationship.
4. Click **Update** when finished.

Alternatively, if you just want to specify the order in which elements in a list box appear see step below only select it as a Child value and click **Update**. Skip to Step 12 for more detailed instructions.

Title	<input type="text"/>
Parent database name	* None *
Child database name	A Component(COMPONENT)
<input type="button" value="Update"/> <input type="button" value="Cancel"/>	

Adding a new Allowed Value Type

5. If you want to create a parent-child dependency between *Category* and *Priority* (using a sample Title of "Test"), do so as seen below:

Title	Test
Parent database name	Category(CATEGORY)
Child database name	Priority(PRIORITY)
<input type="button" value="Update"/> <input type="button" value="Cancel"/>	

Sample New Allowed Value Type

6. Click the **Update** button.

The following screen appears:

	Title	Parent DB Name	Child DB Name
	Category		Category
	Disposition		Disposition
	Product		Product
	Status		Status
	longallowedvaluesitlelong		Attachments
	module		Module
	parent		parent
	parent 2		parent2
	platform		Platforms
	severity		Severity
	Module Component	Module	A Component
	Product Platform	Product	Platforms
	Test	Category	Priority
	module child	Module	child
	parent child1	parent	child
	parent child2	parent	child2
	parent2 parent	parent2	parent
	product parent2	Product	parent2

18 records selected

Sample Allowed Value Type

- Now that you have the dependency established, you can specify the details, i.e. what *Priority* will appear in a list box based on a selected *Category*. To do this, click the **List** icon next to the applicable Allowed Value Title. On the following screen click the **Add** icon.

Add Add a new entry to the list

Category Priority Sort Sequence Owner

Allowed Value List

- Select an item from the Category (Parent) list, and an item from the Priority (Child) list. If you wish, you can control the order in which items appear in the list box by entering an integer value in the Sort Sequence text box.

9. Click **Update** and repeat steps 10 and 11 for each Child value you want to appear for each Parent item.

Add entry to the User Defined Field list for Test

CATEGORY: Enhancement

PRIORITY: 0

Sort Sequence:

Owner: * None *

Adding to the Allowed Value List

10. If you want to set up a sort sequence for a field, for example: if you want to specify the order in which the items in *Status* appear, then follow steps 1, 2, and 3 as above. Once you reach the Add a New Allowed Value Type screen, specify a Title and the Child Database Name that correspond to the field you are setting the sort sequence. (For this example, *Status*).

Add a new Allowed Value Type to the database

Title: Status Order

Parent database name: * None *

Child database name: Status(STATUS)

Severity(RELEASE_SEVERITY)

Severity(SEVERITY_LEVEL)

Sort By(SORT)

Sort Order(REPORT_NAME)

State Change Rules(STATUS_CHANGE)

Status(RELEASE_STATUS)

Status(STATUS)

System Object Access(SEcurity_PERMISSION)

System Security Object Summary(SEcurity_KEYS)

Test Case ID(TEST_CASE_ID)

Test Case Location(TEST_CASE_LOCATION)

Copyright © Sesame Technology, 1999 - 2001. All rights reserved.
Licensed to Sesame Technology
DEV environment - version 3.1.2.1 DEV
Report problems and request enhancements at the [ExtraView support site](#).

Adding a Sort Sequence to Status

11. After clicking on **Update**, you are returned to the Allowed Value Types screen. Click the **List** icon next to the Allowed Value title you want the sort sequence set on, in this case *Status Order*.

Allowed Value Types

Add Add a new allowed value type to the database

	Title	Parent DB Name	Child DB Name
Edit List	Category		Category
Edit List	Disposition		Disposition
Edit List	Product		Product
Edit List	Status		Status
Edit List	Status Order		Status
Edit List	longallowedvaluestitlelong		Attachments
Edit List	module		Module
Edit List	module		Module
Edit List	parent		parent
Edit List	parent 2		parent2
Edit List	platform		Platforms
Edit List	severity		Severity
Edit List	Module Component	Module	A Component
Edit List	Product Platform	Product	Platforms
Edit List	module child	Module	child
Edit List	parent child1	parent	child
Edit List	parent child2	parent	child2
Edit List	parent2 parent	parent2	parent
Edit List	product parent2	Product	parent2

Allowed Value Types screen

12. Click the **Add** icon on the following screen:

Allowed Value List for - Status Order

Add Add a new entry to the list

Status Sort Sequence Owner

Editing Allowed Value List

13. Select an item from the **Status** dropdown menu.

Add entry to the User Defined Field list for Status Order

STATUS Closed

Sort Sequence Closed

Owner Fixed

Open

Pending

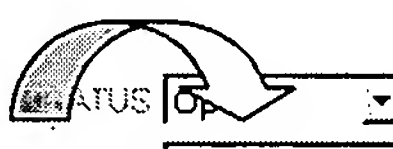

Unassigned

Update Cancel


Selecting the First Sort Item

14. Specify an integer value in the **Sort Sequence** text box.

Add entry to the User Defined Field list for Status Order

 STATUS 

Sort Sequence

Owner  * None *

Setting the Sort Sequence

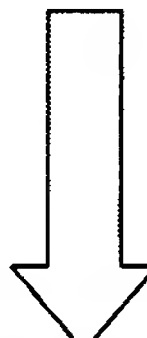
15. Click **Update**, and repeat these three steps until completed.

16. Click **Return**.

Allowed Value List for - Status Order

Add a new entry to the list

	Status	Sort Sequence	Owner
<input type="button" value="Edit"/>	Open	1	
<input type="button" value="Edit"/>	Unassigned	2	
<input type="button" value="Edit"/>	Fixed	3	
<input type="button" value="Edit"/>	Pending	4	
<input type="button" value="Edit"/>	Closed	5	



5 records selected

Completed Sort Sequence List

User and System Wide Reports

These custom reports are generated through the Administration screen or from the Additional reports option on the Search screen. System Wide Reports are reports constructed by an ExtraView Administrator and are available to all of the ExtraView users, while User Reports are created by individual users, for personal queries.


Change a report


Report name ROBBIE.LLOYD.STATUS

Report title

Sort Order

You customize this report by adding columns to a layout you create below. Add columns from the database in the order you want them to appear on the report.

 Add a new column to the report

Column Name	Column Heading	Sequence #
 STATUS	STATUS	1

Change a Report Screen

To create User or System Wide Reports do the following:

1. From the Search/Report screen click the **Additional Reports** button.
2. From the Administration screen click either the **User** or **System Wide Reports** button.
3. To View any of the existing reports just click the **View** button next to the report.
4. To create your own report click the **Add a new personal report** button.
5. Fill in Report Name and Report Title. When adding the report name do not add any special characters or spaces, this is only the database name so it will not be viewable, the report title can contain spaces or special characters.
6. Select the **Sort Order** keys for your report and press update.
7. Next you must add columns to your report by pressing the **Edit** button next to the report you just created, proceed to add as many columns as you please.

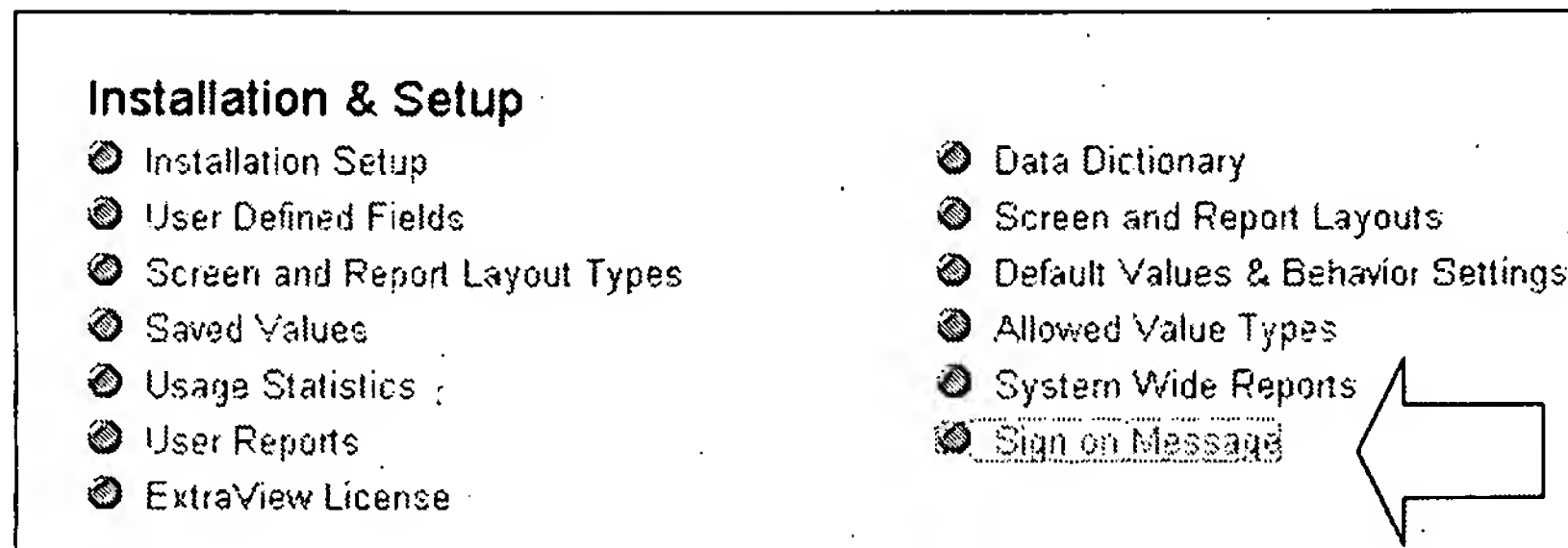
To view this report:

1. From the Search Report screen click Additional Reports and then click the View button next to the report you just created.

Note: A system wide or user report without columns will not appear on your Additional Reports Page until you add columns. It will appear in the Sort Order drop down box so you are able to run Summary, Detailed, or Quick list reports on these sorts.

Sign On Message

Many organizations use the Sign on Message screen as a bulletin board for their organization (system-wide messaging). You are able to use all the conventions of HTML to customize your own Sign on Message.



To create your company's personalized Sign on Message:

1. From the administration menu, under Installation and Setup, click on **Sign on Message**.

The following screen appears:

A screenshot of the 'Sign on Message' screen. The title 'Sign on Message' is at the top. Below it, a text area contains HTML code: `<center>Welcome to ExtraView
<p>View Doc
 Here</p></center>`. A large white arrow points to the text area. Below the text area are two buttons: 'Update' and 'Reset'. At the bottom center is a 'Return' button.

Sign On Message screen

2. You can modify the Sign on Message in any manner. This screen allows you to display HTML on the home screen at the top of the page. When you are finished press the **Update** button.

Below is an example of a Home screen with the Sign on Message:

ExtraView Home

Welcome to ExtraView

View Doc Here

Click on a view icon to see problems assigned to you or choose an option from the menu. Return here at any time using the Home button.

Problems you own:

Status	View	Count	P0	P1	P2	Other
Unassigned		2			1	1

Problems you originated:

Status	View	Count	P0	P1	P2	Other
Unassigned		3			1	2

Use this quick search for simple queries. For more complex searching use the Search/Report menu button.

Status: Priority:
Assigned To: Owner:

Home Page screen

Behavior Settings

The following topics will help you control behavior within ExtraView in order to better suit your company.

Default Values & Behavior Settings (Application Defaults)

This screen gives you the option of editing existing Default Values & Behavior Settings. New values cannot be added from this screen; the specified user is only allowed to change the Value and the Description.
























1. Default Values & Behavior Settings are found under the Installation & Setup sub-heading on the Administration menu.

Installation & Setup

- ☐ Installation Setup
- ☐ User Defined Fields
- ☐ Screen and Report Layout Types
- ☐ Saved Values
- ☐ Usage Statistics
- ☐ User Reports
- ☐ ExtraView License
- ☐ Data Dictionary
- ☐ Screen and Report Layouts
- ☒ **Default Values & Behavior Settings**
- ☐ Allowed Value Types
- ☐ System Wide Reports
- ☐ Sign on Message

Installation & Setup Section

- Clicking on Default Values & Behavior Settings brings up screen similar to the following:

<i>Installation Setup - Default Values & Behavior Settings</i>			
	<i>Database field name</i>	<i>Value</i>	<i>Description</i>
	DEFAULT_USER_GROUP	GUEST	
	IGNORE_USER_GROUP	GUEST	This user group doesn't own and cannot be assigned problems
	DEFAULT_STATUS	UNASSIGNED	Default Status if none visible
	DEFAULT_CATEGORY	SOFTWARE	Default category in the add problem screen
	DEFAULT_PRIVACY	PRIVATE	Default privacy setting when adding problems
	STATUS_CLOSED_NAME	CLOSED	Name of the closed status of a problem.
	DEFAULT_TEXT_REPORT_DELIMITER :		Single character to place between data fields
	FILTER_MODULE_BY_CATEGORY	NO	Allow filtering of modules by category (YES/NO)
	LINK_MODULE_OWNER_ASSIGNED_TO	Yes	Link the module owner field to the assigned to field
	ENFORCE_STATE_CHANGE_RULES	YES	
	ENABLE_PRIVACY_GROUPS	YES	Enables Privacy Group based data security.
	SEPARATE_WORK_FLOW	NONE	Allow separate work flows per USERGROUP or PRODUCT or NONE
	ADMIN_BYPASS_GROUP	ADMIN	Name of group with state change override privileges.
	REPORT_COUNT_METHOD	PROBLEM	PROBLEM, MODULE or RELEASE
	STATUS_TRACKING_METHOD	PROBLEM	PROBLEM or RELEASE
	ALLOW_UNLIMITED_SEARCH	NO	Allows unlimited rows to be returned on queries
	LIMIT_WORD_RECORDS	100	Max number of records returned from a search to a MS Word report
	LIMIT_WORD_DETAILED_RECORDS	25	Max number of detailed records on a MS Word report
	ABRREVIATED_HOME_PAGE	YES	Display the searchable home page or the abbreviated version
	SORT_PRODUCT		
	SORT_MODULE		
	SORT_UDF		
	SORT_SEVERITY		

Default Values & Behavior Settings screen

- Clicking the **Edit** icon next to an item allows you to make changes to the default settings. For example, this is the screen for editing IGNORE_USER_GROUP:
- For each Application Default you want to edit just change the Description and or Value and press **update**.

<i>Change entry in the installation details table</i>	
Fixed database name	IGNORE_USER_GROUP
Description	This user group doesn't own and cannot be assigned problems
Value	GUEST
<input type="button" value="Update"/> <input type="button" value="Cancel"/>	

Editing a Default Value

The following table includes all of the Application Defaults that are editable by the System Administrator, organized by Application Default Name, a sample Value and a Description.

Application Default	Value	Description
DEFAULT_USER_GROUP	GUEST	User Group for User Self-Registration.
IGNORE_USER_GROUP	GUEST	This user group doesn't own and cannot be assigned problems
DEFAULT_STATUS	UNASSIGNED	Default <i>Status</i> if none visible
DEFAULT_CATEGORY	SOFTWARE	Default category in the add problem screen
DEFAULT_PRIVACY	Private	Default privacy setting when adding problems
STATUS_CLOSED_NAME	Closed	This feature allows users to specify what their nomenclature for Closed is. This is vital for using the ALLOWED_EDIT_CLOSED feature.
DEFAULT_TEXT_REPORT_DELIMITER	:	Single character to place between data fields
FILTER_MODULE_BY_CATEGORY	NO	If you would like your <i>Modules</i> to be based on particular <i>Categories</i> as well as <i>Products</i> set this value to YES. When you add <i>Modules</i> in the Configuration section, you will have the opportunity to associate the <i>Module</i> with a particular <i>Category</i> .
LINK_MODULE_OWNER_ASSIGNED_TO	YES	Automatically populates Assigned to field with the appropriate Module Owner.
ENFORCE_STATE_CHANGE_RULES	NO	Ability to turn these rules on or off.
ENABLE_PRIVACY_GROUPS	YES	This turns on the functionality to utilize Privacy Groups (see the section on Security).
SEPARATE_WORKFLOW	USERGROUP	This allows you to enforce State Change Rules based on PRODUCT, USERGROUPS or NONE.
ADMIN_BYPASS_GROUP	ADMIN	The Group to bypass State Change Rules.
REPORT_COUNT_METHOD	RELEASE	You can keep track of ExtraView issues by PROBLEM, MODULE or RELEASE
STATUS_TRACKING_METHOD	RELEASE	You can keep track of the status of ExtraView issues by PROBLEM, MODULE or RELEASE
ALLOW_UNLIMITED_SEARCH	YES	Allows unlimited rows to be returned on queries
LIMIT_WORD_RECORDS	100	The maximum number of MS Word Records that can be outputted for reporting purposes.
LIMIT_WORD_DETAILED_RECORDS	25	The maximum number of MS Word detailed reports that can be returned.
ABBREVIATED_HOME_PAGE	YES	This indicates whether you have a standard home page (YES) or a Customized Home Page (NO).
USER_SELF_REGISTRATION	NO	Allow users to register themselves from the sign on screen
ENFORCE_DETAILED_USER_INFO	YES	By being set to YES this makes nearly all of the fields on the User Registration screen required.
STRICT_PASSWORDS	YES	Require passwords to be at least 6 chars with 1 non-alpha char.
SESSION_EXPIRE_TIME_HOURS	24	Idle time before a user is forced to sign back on to the system.
PASSWORD_EXPIRE_TIME_DAYS	10	Days before a user has to establish a new password.

SUPPORT_LINK	support@sesame.com	Simple HTML used to show the support link at the bottom of each page.
USER_LIST_SIZE	25	This field holds the value for the number of users in a selection list before it is broken down and grouped by letter, user group or name.
EMAIL_NOTIFICATION	YES	This applies to Pop Up text boxes and to the User Accounts screen.
EMAIL_DIRECTORY	/oracle/ExtraView/mailbox	Turn on Email Notification of changes (YES/NO)
EMAIL_FROM_USER_ID	ExtraView.dev@sesame.com	Email Directory for outgoing messages to be stored
EMAIL_FROM_USER_NAME	ExtraView	Return address for all email
EMAIL_MODULE_OWNER_ALWAYS	Yes	Alias for the real user name that email originates from
EMAIL_SUBJECT_TEMPLATE	[\$\$ID\$\$] \$\$\$SHORT_DESC\$\$	Option to send email to the Module Owner.
DEFAULT_USER_FOR_EMAIL	SYSTEM	This allows users to customize their email subject line. The email section has detailed instructions on how to alter this.
OWNER.TYPE	TEXT	User to receive email when a problem is not assigned to anyone
ASSIGNED_TO.TYPE	TEXT	Allows the value to be accessible through a list or a pop up text box.
RELEASE_FOUND.TYPE	TEXT	Allows the value to be accessible through a list or a pop up text box.
RELEASE_FIXED.TYPE	TEXT	Allows the value to be accessible through a list or a pop up text box.

Some of the above Application Defaults that are commonly altered include:

- **DEFAULT_STATUS** – You can set this to any status that you have in your status list
- **DEFAULT_CATEGORY** - You can set this to any category that you have.
- **DEFAULT_PRODUCT** - You can set this to any product that you have.
- **DEFAULT_PRIVACY** - You can set this to either Private or Public.
- **ENFORCE_STATE_CHANGE_RULES** – Gives you the opportunity to turn State Change Rules on or off.
- **USER_SELF_REGISTRATION** – You can allow your users to self-register, or you can be the registrar.
- **SUPPORT_LINK** – If you have an HTML page that you would like your users to go to you can put the link here.
- **EMAIL_FROM_USER_ID** – for example; *ExtraView@yourcompany.com*
- **EMAIL_SUBJECT_TEMPLATE** – This allows you to use fields within the record and place them within the subject line of the email that is generated. The field that you want to include is enclosed between \$\$ and \$\$\$. For example, the problem title is \$\$\$SHORT_DESCR\$\$\$.
- **USERNAME_DISPLAY** – You have the ability to display usernames by First, Last or by ID based on your company's methodology.

Viewing Usage Statistics

In order to efficiently conduct business, vital information must be recorded and analyzed with the intention of perfecting the development process. Usage Statistics can be accessed via the Administration menu, under the Installation & Setup heading.

1. Clicking on **Usage Statistics** brings up the following page:

<i>Usage Statistics</i>	
Report prepared August 17, 2001 11:08	
User Statistics	
Number of users	473
Number of active users	470
Number of active customer users	9
Number of internal users	461
Number of active users in group <i>Administrator</i>	38
Number of active users in group <i>Development Manager</i>	11
Number of active users in group <i>Engineer</i>	441
Number of active users in group <i>Guest</i>	14
Number of active users in group <i>Management</i>	1
Number of active users in group <i>Zephyr Contracting</i>	1
Number of active users in group <i>QA Engineers</i>	8
Number of active users in group <i>QA Manager</i>	6
Number of active users in group <i>Release Management</i>	5
Number of active users in group <i>TAC Engineer</i>	5
Number of active users in group <i>TAC Manager</i>	4
Number of active users in group <i>Tech. Pubs. Manager</i>	4
Number of active users in group <i>Technical Writer</i>	2
Number of active users in group <i>Testing</i>	2

The page is divided into 3 main sections: User Statistics, Problem Statistics, and File Attachments.

User Statistics	
Number of users	467
Number of active users	463
Number of active customer users	11
Number of internal users	452
Number of active users in group <i>Administrator</i>	32
Number of active users in group <i>Development Manager</i>	9
Number of active users in group <i>Engineer</i>	429
Number of active users in group <i>Guest</i>	12
Number of active users in group <i>QA Engineer</i>	3
Number of active users in group <i>QA Manager</i>	3
Number of active users in group <i>Release Management</i>	3
Number of active users in group <i>TAC Engineer</i>	3
Number of active users in group <i>TAC Manager</i>	2
Number of active users in group <i>Tech. Pubs. Manager</i>	2
Number of active users in group <i>Technical Writer</i>	2

User Statistics section

Problem Statistics	
Total number of problems in database	384
Number of problems created to date this month	104
Number of problems created last month (May)	26
Number of problems created in last 30 days	120
Number of updates applied to date this month	506
Number of updates applied last month (May)	150
Number of updates applied in last 30 days	624

Problem Statistics section

File Attachments	
Number of attachments in system	48
Average size of an attachment (kB)	557
Maximum size of an attachment (kB)	10199

File Attachments section

ExtraView Configuration

Configuration items are fields other than UDFs that appear in your installation of ExtraView. The Configuration section of Administration menu will vary

drastically based on different companies' particular implementations.

Altering Configuration Settings



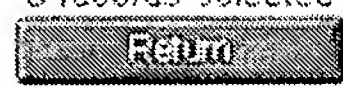
1. From the Administration menu, see the items under Configuration.

Configuration	
● Product Lines	● Products
● Product Releases	● Module Names
● Module Types	● Priorities
● Severity Levels	● Categories
● Statuses	● Resolutions
● Enclosure Types	● Manage Interest Lists

Configuration section

2. For purposes of explanation, we will use Module Names as our example. *Module Names* is a setting that is dependent on other factors (parent-child relationships). For this reason the drop down format is used and the process is slightly different than entering things that are independent.
3. To add or modify a *Module*, click **Module Names**.

The following screen appears:

Module Names					
 Add a new entry to the Module table					
Product	" None " 				
<i>Module Name Module Title Owner Module Type Created Last updated</i>					
0 records selected					
					

Module Names screen

4. Click the **Add** icon to create a new Module Name. Since *Module* depends on *Product*, you will have to choose an applicable *Product* from the drop-down list to associate with your new *Module*. Fill in the appropriate fields and click the **Update** button.

Add entry to the Module table

Product: * None *

Module: _____

Module Owner: _____ ?

Module Type: * None *

Title: _____

Update Cancel Module Interest List

Adding a Module Name

- To modify an already existing *Module Name*, return to the Administration menu and select *Module Name*, as in step 4. Select the *Product* that pertains to the *Module* you want to edit.

Module Names

Add Add a new entry to the Module table

Product: NetOp

Module Title Owner Module Type Created Last updated

Agent Environment	rcrook		12-AUG-1999 07:39	27-APR-2001 11:51
Client	luff	Software	10-DEC-1998 12:59	10-DEC-1998 12:59
EMS			02-JUN-2000 15:56	02-JUN-2000 15:56
EMS Inventory			17-JUN-2000 10:47	17-JUN-2000 10:47
JAM Main			08-JUN-2000 14:43	08-JUN-2000 14:43
PM	Michelle Medina	Hardware	05-JUL-2000 10:13	07-MAR-2001 17:01
Port			07-JUN-2000 13:51	07-JUN-2000 13:51
Security			15-JUN-2000 12:30	15-JUN-2000 12:30
Server	luff	Software	10-DEC-1998 10:22	10-DEC-1998 10:22
Shelf Controller Protection			28-JUN-2000 12:20	28-JUN-2000 12:20
Transaction			12-JUL-2000 12:43	12-JUL-2000 12:43

11 records selected

Return

Editing an Existing Module

6. Click the **Edit** button next to the *Module* that you wish to change.

Module Names

Add Add a new entry to the Module table

Product

	Module Name	Module Title	Owner	Module Type	Created	Last updated
Edit	AGENT ENVIRONMENT	Agent Environment	rcook		chel 12-AUG-1999 07:39	robbie floyd 27-APR-2001 11:51
Edit	CLIENT	Client	luft	Software	cbaldwin 10-DEC-1999 12:59	cbaldwin 10-DEC-1999 12:59
Edit	EMS	EMS			celva 02-JUN-2000 15:55	celva 02-JUN-2000 15:55
Edit	EMS INVENTORY	EMS Inventory			kandiah 17-JUN-2000 15:47	kandiah 17-JUN-2000 15:47
Edit	JAM MAIN	JAM Main			dant 08-JUN-2000 14:43	dant 08-JUN-2000 14:43
Edit	PM	PM	Michelle Medina Hardware		kandiah 05-JUL-2000 10:13	michelle 07-MAR-2001 17:01
Edit	PORT	Port			kandiah 07-JUN-2000 13:51	kandiah 07-JUN-2000 13:51
Edit	SECURITY	Security			rlam 15-JUN-2000 12:30	rlam 15-JUN-2000 12:30
Edit	SERVER	Server	luft	Software	cbaldwin 10-DEC-1999 10:22	cbaldwin 10-DEC-1999 10:22
Edit	SHELF CONTROLLER PROTECTION	Shelf Controller Protection			kandiah 29-JUN-2000 12:29	kandiah 28-JUN-2000 12:29
Edit	TRANSACTION	Transaction			katnich 12-JUL-2000 12:43	katnich 12-JUL-2000 12:43

11 records selected

Return

Edit an Existing Module

7. On the following screen, make the desired changes and click the **Update** button.

Change an entry in the Module table

Product

Module

Module Owner ?

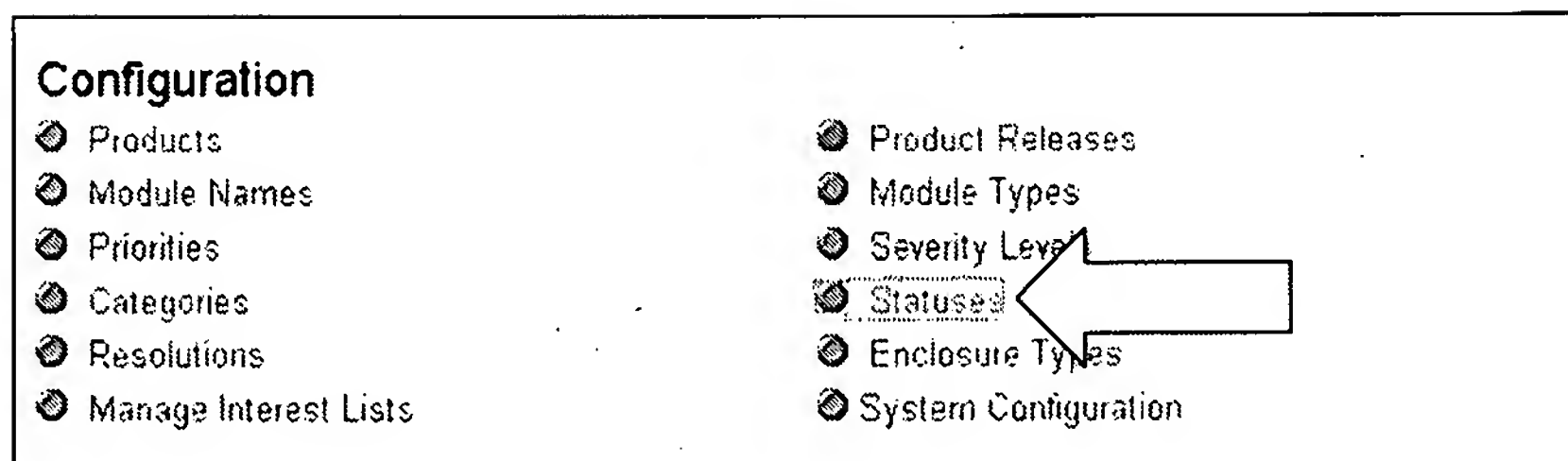
Module Type

Title

Update **Delete** **Cancel** **Module Interest List**

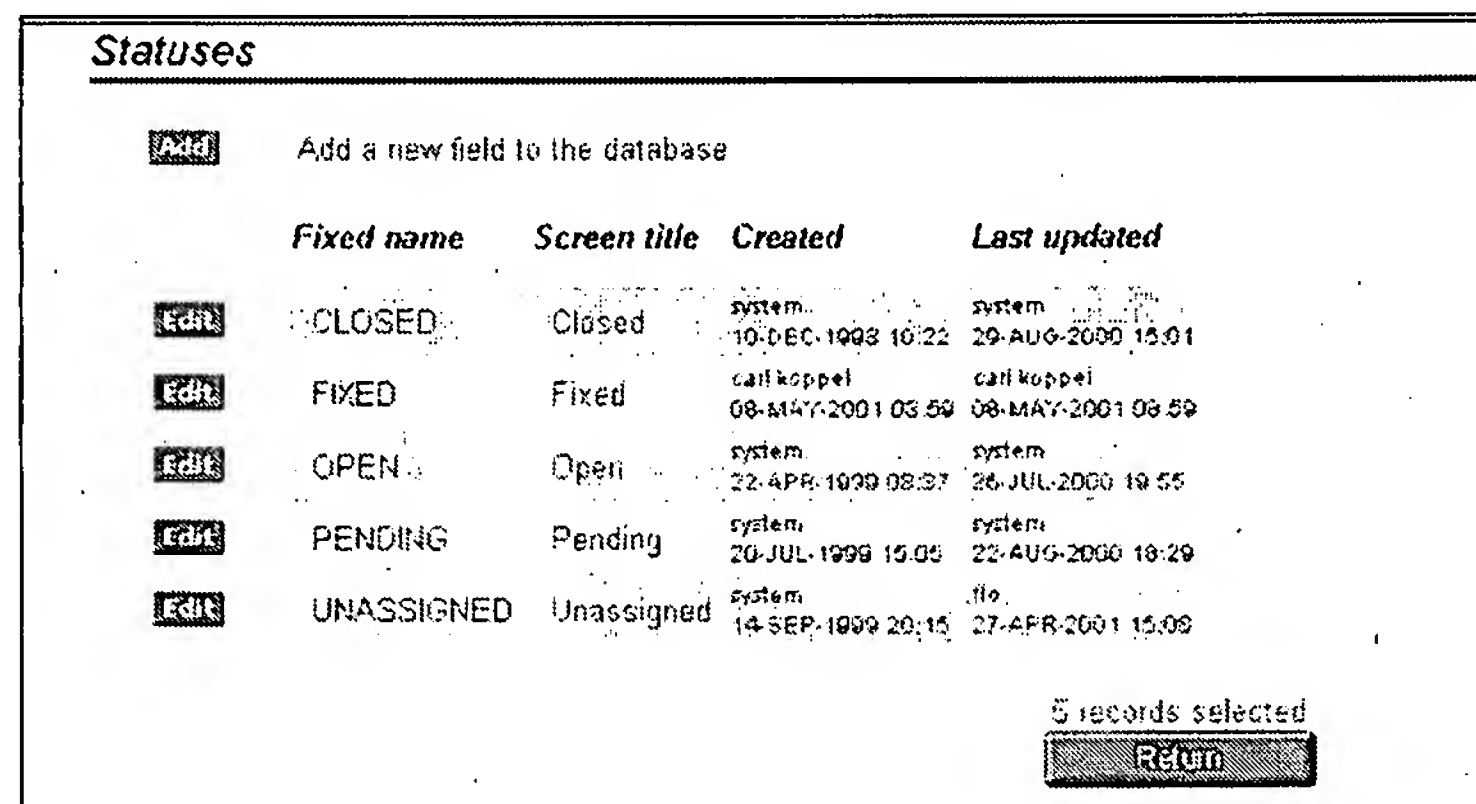
Editing a Module

8. *Statuses* are an example of simple independent lists. To add or modify an independent Configuration Setting, go to the Administration menu, and click the applicable field name in the Configuration section, in this case *Statuses*.



Configuration section

The following Add a new field to the database screen appears:



Statuses screen

9. To add a new status, click the **Add** icon, and fill in the fixed database name and desired display title.

10. Click the **Update** button.

Add a new field to the database

Fixed database name

Title to display on screens

Update **Cancel**

Adding a New Status

11. To edit an existing status, click the **Edit** icon next to the status name you wish to change. You can either change its screen title or delete it. Click the **Update** button when finished.

Change a Status's details

Fixed database name

CLOSED

Title to display on screens

Closed

Update

Delete

Cancel

Editing an Existing Status

User Roles

You have the ability to change your role as long as you belong to more than one User Group. This allows you to wear a "different hat". By changing your role you can test the functionality of your implementation and clearly see the permissions of a particular User Group without having to login repeatedly.

In order to change your user role without logging out, just click on **click to change your current user role** located near the bottom of the home page.

sesame

My Home

Add Problem

Search/Report

Administration

Help

Sign Off

Bug #

ExtraView

Problems you originated:

Status	View	Count	F0	P1	P2	Other
Unassigned		3				

Use this quick search for simple queries. For more complex searching use the Search/Report menu button.

Status

Any

Priority

Any

Assigned To

?

Owner

?

Product

Any

Category

Any

Module

Any

From Date

To Date

Date created

?

Last Modified

?

Version Open

?

Sort By

Priority

Bug ID

Date Modified

Ascending

Descending

Records per Page

20

Search

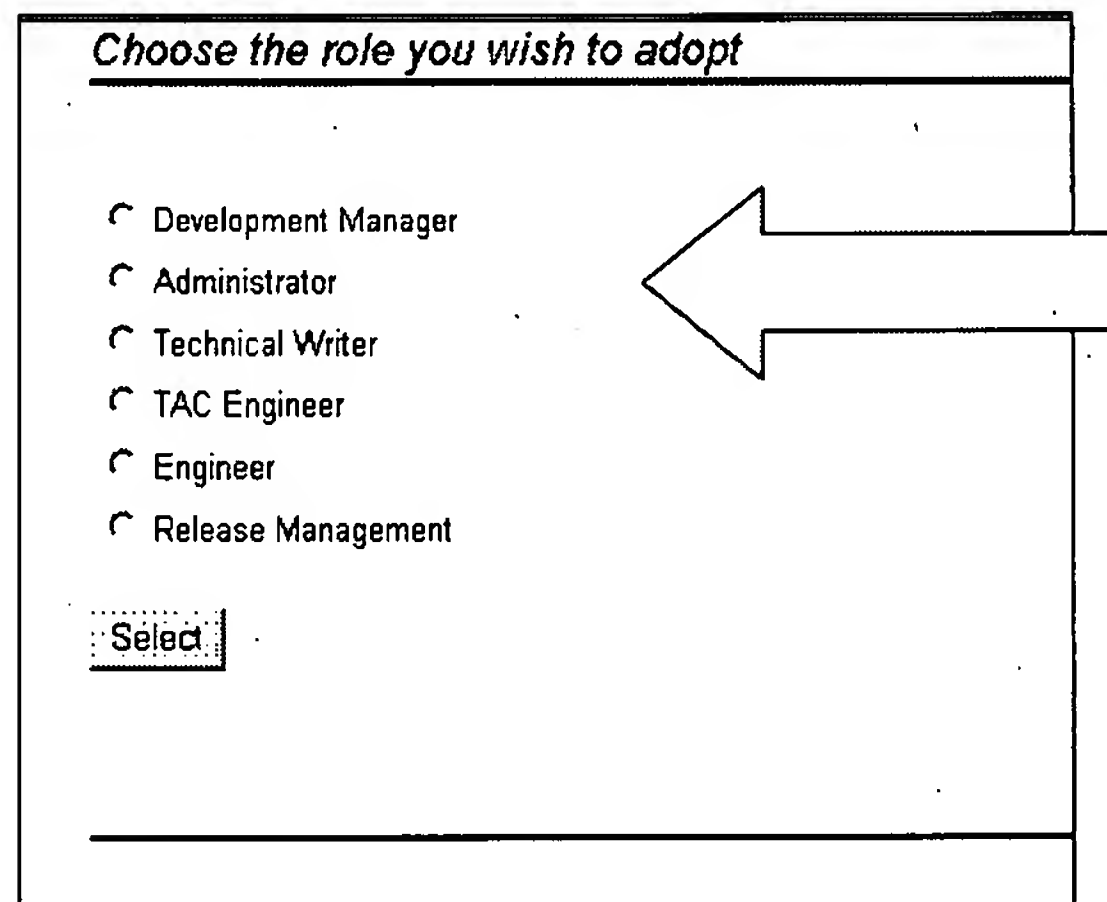
Click to change your current user role.

Click to edit your personal options.

Click to edit your personal email interest list.

Home Page screen

Select the role you would like to play by selecting a radio button from any User Group and then clicking the **select** button. Your role has now been changed and your permissions will reflect the change. You may find that access to certain buttons on the left navigation bar disappears, or that fields no longer show up in the add/edit screens, or that buttons are added and fields appear in some screens; this is a direct result of changing your role.

A screenshot of a web application window titled "Choose the role you wish to adopt". The window contains a list of roles with radio buttons: Development Manager, Administrator, Technical Writer, TAC Engineer, Engineer, and Release Management. Below the list is a "Select" button. A large white arrow points from the right side of the window towards the list of roles.

Choose the role you wish to adopt

☐ Development Manager

☐ Administrator

☐ Technical Writer

☐ TAC Engineer

☐ Engineer

☐ Release Management

Select

User Role screen

Customized ExtraView Home Screen

Different installations may have different customized Home Pages. The main feature of the Home Page shown is a Data Summary Table:

- Provides a list of the user's currently active problems by product
- Provides a list of the issues that the user created
- Provides the status of all currently active problems by product
- Provides the user a drill-down view of all currently active problems by product
- Provides a count of all currently active problems by product

Home Page screen

In addition to the Data Summary Table, the top part of the Home Page has a message field that the system administrator can alter at any time.

Perhaps the most innovative feature of the Home Page is the Quick Search option that is provided. Here you are able to use the majority of the search parameters provided on the Search/Report screen to begin searching. Also note that in Unix, the pick list is not scrollable but static.

If you or your customers are not interested in searching from the Home Screen, you can turn this feature off.

1. From the Administration menu, click on **Default Values and Behavior Settings** and then click **Edit** next to ABBREVIATED_HOME_PAGE. From this screen you are able to enter “yes” or “no” based on your preference.

Problems you originated

Status	View	Count	P0	P1	P2	Other
Unassigned		3			1	2

Use this quick search for simple queries. For more complex searching use the Search/Report menu button.

Status
Priority

Assigned To
Owner

Product

Category

Module

Version Open

From Date To Date

Date created

Last Modified

Sort By
☒ Priority
☐ Bug ID
☐ Date Modified

☒ Ascending
☐ Descending

Records per Page

- Click to change your current user role.
- Click to edit your personal options.
- Click to edit your personal email interest list.

Home Page screen

To use the search option from the home page:

1. Select anything in the fields above to build your query. For example, you can select one option or multiple options from status or from product or both. Then click on the **Search** button and the results are displayed.

Quick List							
Prepared by during on 2001-06-20							
Records: 1 to 20 of 204				<input type="button" value="Next"/>	<input type="button" value="Refresh"/>		
Bug #	Title	Product	Module	Assigned To	Last Modified		
Version Open	Version Closed	Status	Priority	Owner	Days in Queue		
Edit View	12521	calendar	SMS	802.1q	des	2001-06-29 10:16	
18jun99		Unassigned	0			0	
Edit View	12520	Test	Robbie	4		2001-06-29 09:43	
7		Closed	0			0	
Edit View	12519	a	SMS	AAAb	carrel	2001-06-29 09:24	
18jun99		Closed	0			0	
Edit View	12511	Testing Time Zones	Robbie	4		2001-06-22 13:42	
7		Pending	0			6	
Edit View	12492	test original	SMS	AAAb	florence	2001-06-26 15:04	
18jun99		Unassigned	0		florence	8	
Edit View	12490	test refresh button	SMS	AAAb	flo	2001-06-27 12:57	
18jun99		Open	0		flo	8	
Edit View	12489	quest test	Dev Test	Test Automation	flo	2001-06-21 09:29	
1.9		Pending	0		florence	8	
Edit View	12488	add problem as a quest	SMS	802.1q	flo	2001-06-21 09:40	
1.4.1		Pending	0		florence	8	
Edit View	12478	xcs	SMS	802.1q	des	2001-06-21 09:42	
18jun99		Open	0		flo	8	

Quick List

Editing Your Personal Options

Editing personal options allows you control how you want to enter dates into the system, what your password will be, user information that others will see, notification via e-mail, how many records you view per page and what privacy group you belong to.

Problems you originated:

Status	View	Count	P0	P1	P2	Other
Unassigned		3			1	2

Use this quick search for simple queries. For more complex searching use the Search/Report menu button

Status

Assigned To

Product

Category

Module

Version Open

Priority

Owner

From Date

To Date

Date created

Last Modified

Sort By ☒ Priority
☐ Bug ID
☐ Date Modified

☒ Ascending
☐ Descending

Records per Page

Home Page screen

Prior to entering this screen you will be asked to enter your password.

Enter your password

Enter password

Copyright © Sesame Technology, 1999 - 2001. All rights reserved.
Licensed to Sesame Technology
DEV environment - version 3.1.2.1 DEV
Report problems and request enhancements at the [ExtraView support site](#).

Secondary Level Password screen

Change your personal details

User ID

ROBBIE.LLOYD

First name

Robbie

Last name

Lloyd

Password

Verify Password

Update

Cancel

Email address

rlloyd@sesame.com

Email format

HTML

Date format

DD-MON-YYYY hh24:mi

Timezone

GMT -8 hours

Notify on own updates

☒ Yes ☐ No

Records per Page

1000

Job title

Company name

Address

Sesame

City

State / Province

Zip / Post Code

Country

Work phone

Home phone

Cell phone

Fax

Pager

Change Personal Details screen

MANAGE USERS & SECURITY

ExtraView allows the System Administrator to create an unlimited number of user groups and users, and to define, and then apply security or access privileges to users based on these groups. These access privileges are applied to each and every field, to each and every menu button, and various other features within ExtraView, allowing you to control access to nearly every feature of the product.

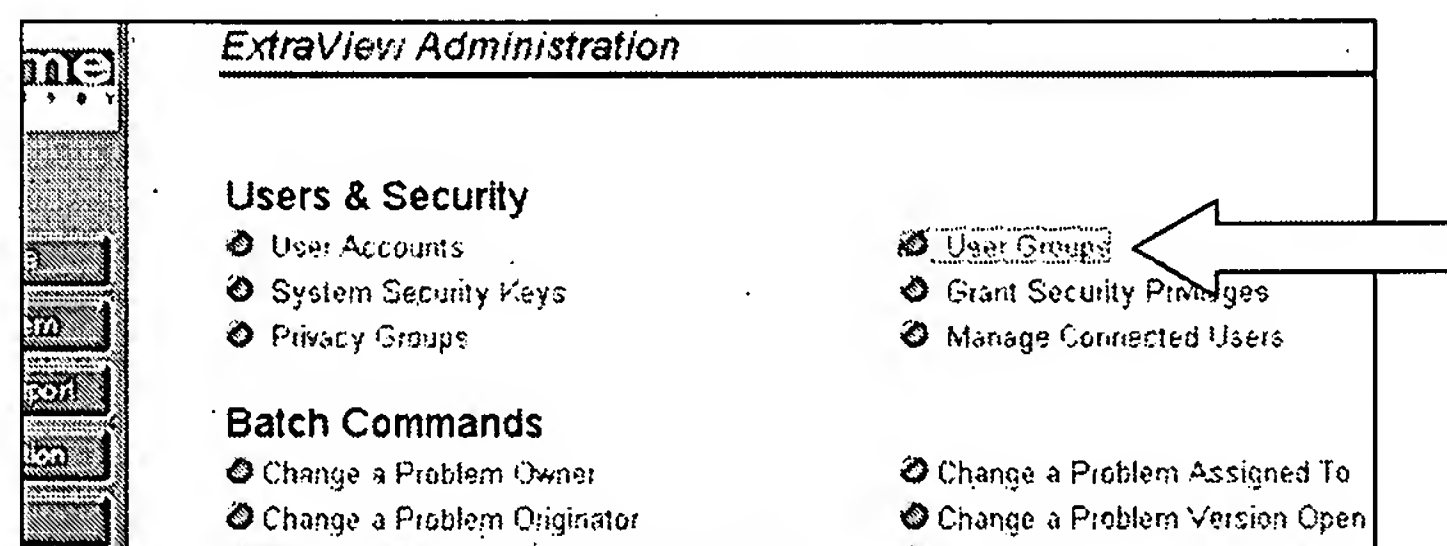
User Groups

User Groups are the functional teams of your company or the external users that will be using ExtraView. User Groups are assigned specific privileges based on what you want each of them to be able to see and do. Sesame Technology will generally provide the initial User Groups setup for you at the time of license. These initial groups may or may not include the following:

- Administrator
- Customer
- Engineer
- QA

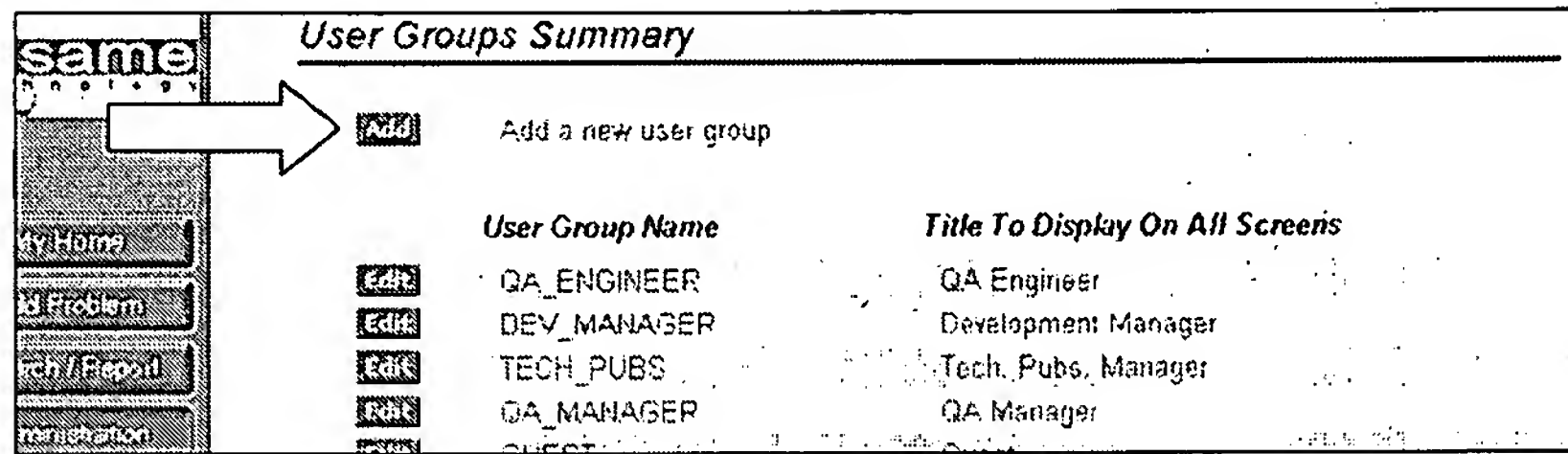
Add User Groups

1. Click **User Groups** under Users & Security on the Administration menu.



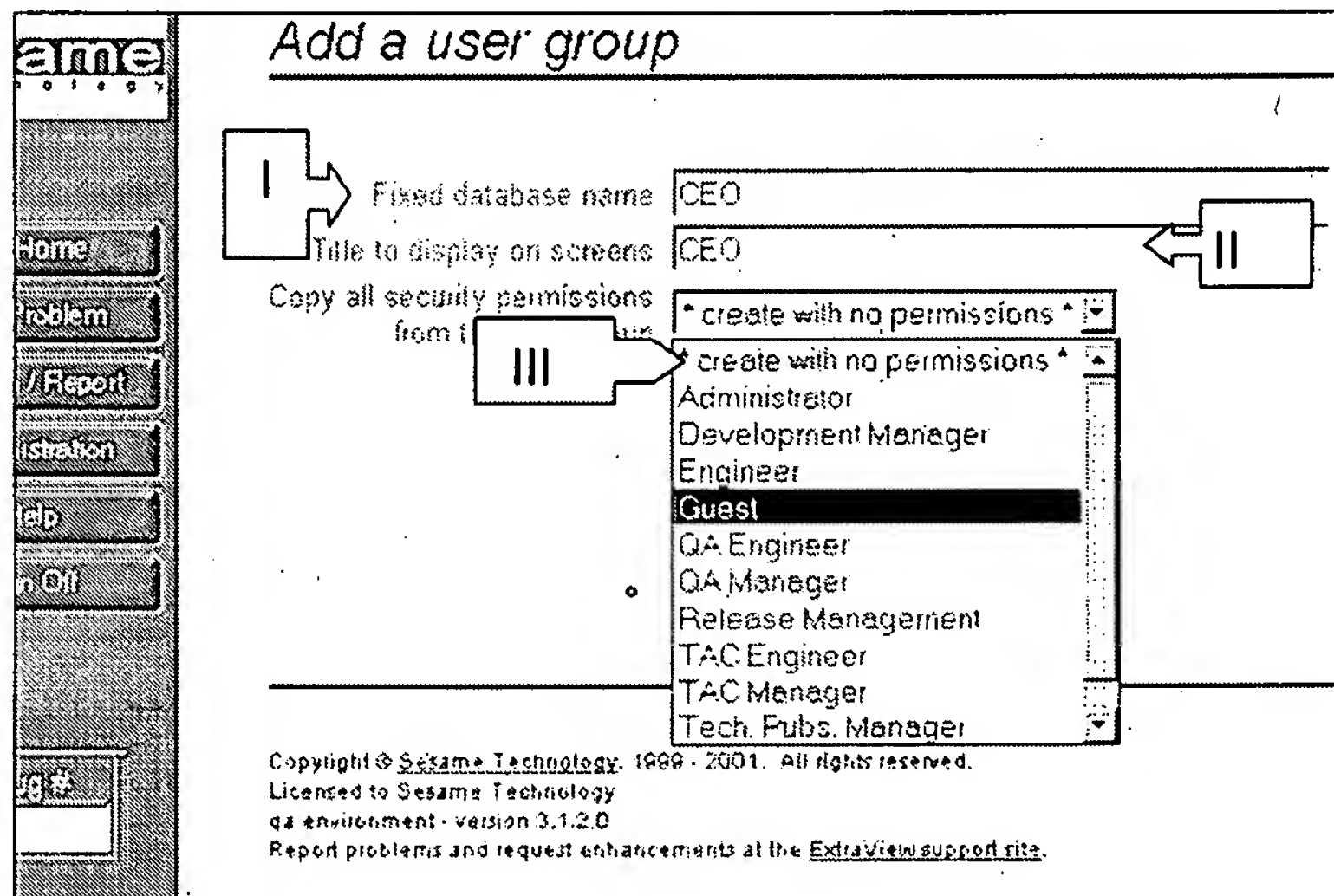
Administration screen

2. Click the **Add** button on the User Groups Summary screen.



User Groups Summary screen

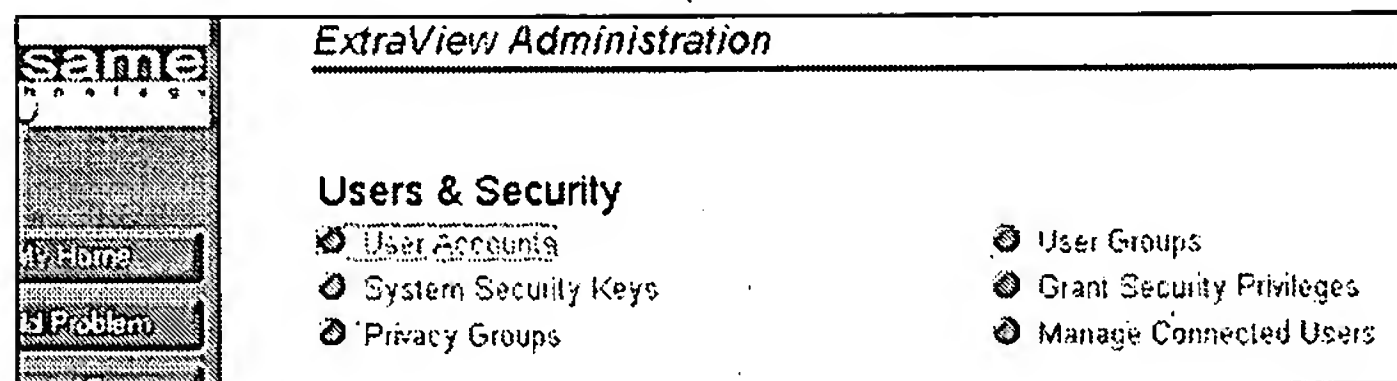
3. Type in a database name, and make it relevant to the name you will use as a title (I). Valid Database names have no spaces, and are composed of letters, numbers, and the characters, -, _, and '+.
4. Type in a title (II).
5. You can then choose to clone the Security Privileges from another existing group, or set your own from Grant Security Privileges by first selecting the *create with no permissions* option (III). Cloning a group is particularly useful when you want to create a new group similar to an existing one. After cloning the privileges, you can make any changes to the new group from the Grant Security Privileges screen.



Add a User Group screen

Add New Users

1. Click **User Accounts** from the Administration menu.



Administration screen

The User Accounts Summary screen appears.

At the top of the User Accounts Summary screen there is the increased functionality to narrow the list of users returned.

- To return inactive users, all users or users belonging to a particular user groups select the appropriate option from the **Select users to display** dropdown
- To return all users with a User ID beginning with a particular letter just click on the appropriate letter. The example below indicates that the letter **F** was selected.
- To search for one particular user, you can type the User ID in the **Search by ID** field and you can press the **Go** button.
- If you would like the entire list to be displayed just change the **Default Value and Behavior Setting** called **USER_LIST_SIZE**. This field holds the value for the number of users in a selection list before it is broken down and grouped by letter, user group or name.

User Accounts Summary

Add a new user Select users to display: All active users

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z Other Search by ID:

	User ID	User Name	Company	Last Access	Created	Last Updated
	FARZAM	farzam	QA	system	15-FEB-2001 15:41	system 15-FEB-2001 15:41
	FLING	fling	QA	system	15-FEB-2001 15:41	system 15-FEB-2001 15:41
	FLO	red white and blue	QA	14-AUG-2001	system 15-FEB-2001 15:41	16-04-JUN-2001 10:59
	FLORENCE	Florence Ng	QA	08-SEP-2000	system 15-FEB-2001 15:41	16-04-JUN-2001 10:59
	FLORENCE SHUETPING	ilo ng	QA	16-APR-2001	16-MAR-2001 11:12	16-MAY-2001 12:45
	FNORD	fnord	QA	system	15-FEB-2001 15:41	system 15-FEB-2001 15:41
	FRANCIS	francis	QA	system	15-FEB-2001 15:41	system 15-FEB-2001 15:41
	FZAENI	fzaeni	QA	system	15-FEB-2001 15:41	system 15-FEB-2001 15:41

8 records selected

User Accounts Summary screen

2. Click the **Add New User** icon. The Add New User screen appears.

Add new user	
User ID <input type="text"/> First name <input type="text"/> Last name <input type="text"/> Password <input type="password"/> Verify Password <input type="password"/> Expire password <input type="checkbox"/> User account <input checked="" type="radio"/> Enabled <input type="radio"/> Disabled User Groups <input type="checkbox"/> Administrator <input type="checkbox"/> Development Manager <input type="checkbox"/> Engineer <input type="checkbox"/> Guest <input type="checkbox"/> QA Engineers <input type="checkbox"/> QA Manager <input type="checkbox"/> TAC Engineer <input type="checkbox"/> TAC Manager Privacy Groups <input type="checkbox"/> ABC Corp. <input type="checkbox"/> Emulator	Email address <input type="text"/> Email Format <input type="text" value="HTML"/> Date format <input type="text" value="YYYY-MM-DD hh24:mi"/> Drilldown Report format <input type="text" value="Quick List"/> Timezone <input type="text" value="(GMT) Greenwich Mean Time London"/> Notify on own updates <input checked="" type="radio"/> Yes <input type="radio"/> No Job title <input type="text"/> Company name <input type="text" value="ExtraView"/> Address <input type="text"/> <input type="text"/> <input type="text"/> City <input type="text"/> State / Province <input type="text"/> Zip / Post Code <input type="text"/> Country <input type="text"/> Work phone <input type="text"/> Home phone <input type="text"/> Cell phone <input type="text"/> Fax <input type="text"/> Pager <input type="text"/>
<input type="button" value="Update"/> <input type="button" value="Cancel"/>	

Add New User screen

Enter New User Information:

User ID	The name that the new user will use to login to the system (required)
First Name	User's first name (required)
Last Name	User's last name (required)
Password	User's password (required)
Verify Password	Retype User's password (required)
User Account	Enabled User may login Disabled User may not login
User Groups	A functional team where members have the same set of privileges
Privacy Groups	Privacy Groups enable certain groups within ExtraView to view different specific sets of issues without making the issue PUBLIC (all users can see it) or PRIVATE (only internal users can see it)
Email Address	The email address to which automatic email notification will be sent
Email Format	HTML, Plain Text (brief), or Plain Text (full). The brief option provides just a few fields, enough to recognize the issue
Date Format	Date format selection applies for the new user system-wide
Time Zone	Time-zone selection applies for the user system-wide
Notify on own updates	Enabled will receive automatic email Disabled will not receive automatic email
Company Name	For internal users, this should reflect the same value as COMPANY_NAME in installation setup
User Profile Fields	Enter the appropriate user profile and contact information

3. After you have finished entering the information, click the **Update** button, and you will be returned to the User Accounts screen, where you can add another user or return to the Administration menu.

Note: If the application default named ENFORCE_DETAILED_USER_INFO is set to "Yes", then additional fields will become required. This is used when you want users that self-register on the system to provide a significant level of personal details.

Edit User Accounts

1. To edit user accounts, click on **User Accounts** from the Administration menu and the User Accounts Summary screen will appear.

At the top of the User Accounts Summary screen there is the increased functionality to narrow the list of users returned.

- To return inactive users, all users or users belonging to a particular user groups select the appropriate option from the **Select users to display** dropdown.
- To return all users with a User ID beginning with a particular letter just click on the appropriate letter. The example below indicates that the letter **F** was selected.
- To search for one particular user, you can type the User ID in the **Search by ID** field and you can press the **Go** button.

Note: The above functionality can be utilized when Editing or Deleting a User Account.

User Accounts Summary

Add

Add a new user
Select users to display
All active users

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z Other
Search by ID
Go

	User ID	User Name	Company	Last Access	Created	Last Updated
Edit	FARZAM	farzam	QA	system	15-FEB-2001 15:41	system 15-FEB-2001 15:41
Edit	FLING	fling	QA	system	15-FEB-2001 15:41	system 15-FEB-2001 15:41
Edit	FLO	red white and blue	QA	14-AUG-2001	system 15-FEB-2001 15:41	file 04-JUN-2001 10:53
Edit	FLORENCE	Florence Ng	QA	08-SEP-2000	system 15-FEB-2001 15:41	file 27-APR-2001 09:11
Edit	FLORENCE SHUETPING	ilo ng	QA	10-APR-2001	file 15-MAR-2001 11:12	file 20-MAY-2001 12:40
Edit	FNORD	fnord	QA	system	15-FEB-2001 15:41	system 15-FEB-2001 15:41
Edit	FRANCIS	francis	QA	system	15-FEB-2001 15:41	system 15-FEB-2001 15:41
Edit	FZAENI	fzaeni	QA	system	15-FEB-2001 15:41	system 15-FEB-2001 15:41

6 records selected
Return

User Accounts Summary Screen

2. Click the **Edit** button beside the user account you wish to edit.

The user information screen then appears.

3. When you are done changing the desired information, click **Update**.

Change a user's details

User ID

ROBBIE.LLOYD

Email address

rlloyd@sesame.com

First name

Robbie

Email format

HTML

Last name

Lloyd

Date format

DD-MON-YYYY hh24:mi

Password

Drilldown Report format

Detailed Report

Verify Password

Timezone

(GMT -4:00) Atlantic Time(US & Canada)

Expire password

☐

Notify on own updates

☒ Yes ☐ No

User account

☒ Enabled ☐ Disabled

Job title

User groups

☒ Administrator
☐ Development Manager
☒ Engineer
☐ Guest
☒ QA Engineers
☒ QA Manager
☐ TAC Engineer
☐ TAC Manager

Company name

ExtraView

Privacy Groups

☒ ABC Corp.
☐ Emulator

Address

123 Main Street

City

Pleasantville

State / Province

IL

Zip / Post Code

12345

Country

USA

Work telephone

Home telephone

Cell phone

Fax

Pager

Update

Delete

Cancel









Change User Details

Delete User Accounts

1. To delete user accounts, click on User Accounts from the Administration menu, and then click the edit button beside the user account you wish to delete. The user information screen then appears.
2. To disable a user's login access, click the **Disable** User Account radio button, and then click **Update**.
3. To completely remove a user from the system, click the **Delete** button, and then click **Update**. Note that a user can only be removed if he or she no longer has any issues associated with their User ID. You may wish to make the user inactive if he or she is no longer authorized to use ExtraView, but still has issues in the system.

System Security Keys

Each field, button and feature within ExtraView is controlled by what is known as a System Security Key. From the Grant Security Privileges section you are able to manipulate the Read and Write privileges for these keys to allow or restrict access to these fields, buttons, and features.

System Security Object Summary		
	Add a new security key	
	Security Key Name	Title to display on Screens
	CF_ALLOWEDVALUE_TYPE	Allowed Value Types
	CF_ALLOWED_VALUES	Allowed Value List
	CF_BATCH_COMMANDS	Batch Commands
	CF_BUSINESS_RULES	Business Rules
	CF_CATEGORY	Categories
	CF_DATA_DICTIONARY	Data Dictionary
	CF_DEFAULT_VALUES	Default Values & Behavior Settings

System Security Keys Summary

Edit an Existing Security Key

1. Click on the **System Security Key** option on the ExtraView Administration menu.
2. Click on **Edit** and change the description or title as needed.

Add a New Security Key

1. To make a new key, click on **Add a New Security Key**.

Note: This should rarely, if ever, need to be done, as new security keys are added automatically when you create new fields within ExtraView.

2. Enter a name and a key is created (I). Database names have no spaces, and are composed of letters, numbers, and the characters -, _, and +.
3. Title the new key as you wish (II)
4. Click the **Update** button (III).

Add a New Security Key screen

Grant Security Permissions

Granting Security Permissions controls group member's access to all fields, buttons and features in ExtraView. In setting permissions, the System Administrator has the ability to make these kinds of items read only, write only, both readable and writable, or else invisible.

Edit Security Privileges

1. Click **Grant Security Privileges** from the Administration menu.

Administration screen

The Grant Security Privileges screen appears.

Grant Security Privileges

- Security permissions for adding records
- Security permissions for editing records and querying
- Security permissions for administration options
- Security permissions for accessing screens
- Security permissions for security access
- Security permissions for statuses

Grant Security Privileges screen

This screen gives you access to all of the System Security Keys. The options are as follows:

- Security permissions for adding records (Add Problem Items)
- Security permissions for editing records and querying (Edit and Search Screen Items)
- Security permissions for administration options (Administration Options)
- Security permissions for accessing screens (Screen Access Options)
- Security permissions for security access (Security Values within the Status Field on Both the Add and Edit Screens)
- Security permissions for statuses (Status Values within the Status Field on Add and Edit Screens)

By clicking on one of the above options, the administrator has the ability to set read and write privileges for any function in the entire system.

Grant Security Privileges - Adding Records						
	Administrator		Development Manager		Engineer	Guest
Alternate ID Alternate ID for this issue. PR_ADD_PROBLEM.ALT_ID	Read <input checked="" type="checkbox"/> Write <input checked="" type="checkbox"/>	Read <input type="checkbox"/> Write <input type="checkbox"/>	Read <input type="checkbox"/> Write <input type="checkbox"/>	Read <input type="checkbox"/> Write <input type="checkbox"/>	Read <input type="checkbox"/> Write <input type="checkbox"/>	Read <input type="checkbox"/> Write <input type="checkbox"/>
Assigned To Assigned To PR_ADD_PROBLEM.ASSIGNED_TO	Read <input checked="" type="checkbox"/> Write <input checked="" type="checkbox"/>	Read <input checked="" type="checkbox"/> Write <input checked="" type="checkbox"/>	Read <input checked="" type="checkbox"/> Write <input checked="" type="checkbox"/>	Read <input type="checkbox"/> Write <input type="checkbox"/>	Read <input type="checkbox"/> Write <input type="checkbox"/>	Read <input type="checkbox"/> Write <input type="checkbox"/>
Attachment Attachment PR_ADD_PROBLEM.ATTACHMENT	Read <input checked="" type="checkbox"/> Write <input checked="" type="checkbox"/>	Read <input checked="" type="checkbox"/> Write <input checked="" type="checkbox"/>	Read <input checked="" type="checkbox"/> Write <input checked="" type="checkbox"/>	Read <input type="checkbox"/> Write <input type="checkbox"/>	Read <input type="checkbox"/> Write <input type="checkbox"/>	Read <input type="checkbox"/> Write <input type="checkbox"/>
Category Problem category. PR_ADD_PROBLEM.CATEGORY	Read <input checked="" type="checkbox"/> Write <input checked="" type="checkbox"/>	Read <input checked="" type="checkbox"/> Write <input checked="" type="checkbox"/>	Read <input checked="" type="checkbox"/> Write <input checked="" type="checkbox"/>	Read <input checked="" type="checkbox"/> Write <input checked="" type="checkbox"/>	Read <input checked="" type="checkbox"/> Write <input checked="" type="checkbox"/>	Read <input checked="" type="checkbox"/> Write <input checked="" type="checkbox"/>
CC Email PR_ADD_PROBLEM.CC_EMAIL	Read <input checked="" type="checkbox"/> Write <input checked="" type="checkbox"/>	Read <input checked="" type="checkbox"/> Write <input checked="" type="checkbox"/>	Read <input checked="" type="checkbox"/> Write <input checked="" type="checkbox"/>	Read <input type="checkbox"/> Write <input type="checkbox"/>	Read <input type="checkbox"/> Write <input type="checkbox"/>	Read <input type="checkbox"/> Write <input type="checkbox"/>
Clarify ID This is the number of the Clarify problem PR_ADD_PROBLEM.CLARIFY_ID	Read <input checked="" type="checkbox"/> Write <input checked="" type="checkbox"/>	Read <input checked="" type="checkbox"/> Write <input checked="" type="checkbox"/>	Read <input checked="" type="checkbox"/> Write <input checked="" type="checkbox"/>	Read <input checked="" type="checkbox"/> Write <input checked="" type="checkbox"/>	Read <input checked="" type="checkbox"/> Write <input checked="" type="checkbox"/>	Read <input checked="" type="checkbox"/> Write <input checked="" type="checkbox"/>
Comments test PR_ADD_PROBLEM.COMMENTS	Read <input checked="" type="checkbox"/> Write <input checked="" type="checkbox"/>	Read <input checked="" type="checkbox"/> Write <input checked="" type="checkbox"/>	Read <input checked="" type="checkbox"/> Write <input checked="" type="checkbox"/>	Read <input checked="" type="checkbox"/> Write <input type="checkbox"/>	Read <input checked="" type="checkbox"/> Write <input checked="" type="checkbox"/>	Read <input checked="" type="checkbox"/> Write <input checked="" type="checkbox"/>

Grant Security Privileges Screen

Make a Field or Option Read Only

1. Identify the column head for the User Group whose privileges you would like to change.
2. Put a check mark in the **Read** box for that User Group only.
3. Click the **Update** button.

This change will make the field that corresponds to the given security key read only for the selected user group.

Make a Field or Option Write Only

1. Identify the column head for the User Group whose privileges you would like to change.

2. Put a check mark in the **Write** box, for that User Group only.
3. Click the **Update** button.

This change will make the field that corresponds to the given security key writable for the selected user group.

Make a Field or Option Read and Write for a Particular User Group

1. Identify the column head for the User Group whose privileges you would like to change.
2. Put check marks in both of the boxes, for that User Group only (Read and Write).
3. Click the **Update** button.

Make a Field Invisible to a Particular User Group

1. Identify the column head for the User Group whose privileges you would like to change.
2. Remove the check marks from both of the boxes for that User Group (Read and Write).
3. Click the **Update** button.

Company Name Security

There is an application default in Administration under Installation & Setup that is entitled **COMPANY_NAME**. ExtraView recognizes this as the main company name. Given that this is the case, a new user added with a company name that is different than the value in the field associated with **COMPANY_NAME** will not be able to see **PRIVATE** issues submitted by users in your company. This is especially beneficial if you have customers using the system, and you only want them to be able to add issues.

Add new user

User ID

First name

Last name

Password

Verify Password

Expire password

User account

User Groups

Privacy Groups

Update

Cancel

Email address

Email Format

Date format

Drilldown Report format

Timezone

Notify on own updates

Job title

Company name

Address

City

State / Province

Zip / Post Code

Country

Work phone

Home phone

Cell phone

Fax

Pager

Add a New User screen

If you have users/customers who self-register, they will be able to enter your company name (COMPANY_NAME), but they will automatically be assigned to the user group entered as the DEFAULT_USER_GROUP. The DEFAULT_USER_GROUP is set in the Default Values and Behavior Settings. This group normally has minimal user privileges. By being added to this group, the user will only be able to view PUBLIC records until a System Administrator re-assigns them to one or more User Groups.

Manage Connected Users

This feature gives the System Administrator complete knowledge of who is on the system at any time. The System Administrator is also permitted to sign off any inactive users who may have forgotten to sign off, in order to let other users access the system. system by selecting a user and pressing the *Update* button.

63

Manage Connected Users

The following users are currently signed on to ExtraView. Please check the users you want to disconnect, to release licenses back to the common pool. This installation is licensed for 10 users.

Currently Connected	User ID	User Name	License Expiry Time
<input checked="" type="checkbox"/>	flo	red white and blue	04-18-2001 15:22
<input type="checkbox"/>	flo	red white and blue	04-18-2001 11:43
<input type="checkbox"/>	robbie.lloyd	Robbie Lloyd	19-Apr-2001 10:34

3 records selected

Manage Connected Users screen

This feature may or may not appear, depending on the type of licensing that a user possesses. If you have concurrent licensing, you will receive this feature, and you will be able to allow only a limited number of persons to access the system.

Disconnect Inactive Users

1. Click **Manage Connected Users** on the Administration menu.
2. Click on the checkbox associated with the user(s) whom you want to remove from the system (I).
3. Click the **Update** button (II). This action will force the selected user sessions to expire and open up concurrent license seats.

Manage Connected Users

The following users are currently signed on to ExtraView. Please check the users installation is licensed for 10 users.

Currently Connected	User ID	User Name	License Expiry Time
<input checked="" type="checkbox"/>	flo	red white and blue	04-18-2001 15:22
<input type="checkbox"/>	flo	red white and blue	04-18-2001 11:43
<input type="checkbox"/>	robbie.lloyd	Robbie Lloyd	19-Apr-2001 10:34

Diagram annotations: A box labeled 'I' with an arrow points to the first checkbox. A box labeled 'II' with an arrow points to the 'Update' button.

Manage Connected Users screen

Privacy Groups

Privacy Groups are a feature that allows the System Administrator to divide information and data in the system so that select users can be excluded from access to certain problems (something that the PRIVATE designation does not provide). Also, this functionality differs from controlling access through read and write Security Privileges by

virtue of the fact that Privacy Groups do not control access by limiting field-level views of data records, but rather limit access to the data record itself.

PRIVACY_GROUP

Add a new privacy group

Privacy Group Name	Title To Display On All Screens
<input type="button" value="Edit"/> TEST	Test
<input type="button" value="Edit"/> GROUP_XYZ	group xyz

2 records selected

Privacy Groups

For example, if I belong to the Test Privacy Group, then anyone in that privacy group can see my problems, but no one outside the privacy group can do so (unless the value in COMPANY_NAME is the same as the host company name). Additionally, as a member of the Test Privacy Group only, I cannot see any problems that have been submitted by members of the XYZ Privacy Group. The only users that are allowed access to these issues are the members of the host company that have the COMPANY_NAME value, and any other user whom belongs to the XYZ Privacy Group. Finally, if the problem is made PUBLIC instead of PRIVATE, then everyone who uses the system can see the record, regardless of the Privacy Group to which they belong.

Add a Privacy Group

1. Click on **Privacy Groups** under Users & Security on the Administration menu.

ExtraView Administration

Users & Security

- ☒ User Accounts
- ☒ System Security Keys
- ☒ **Privacy Groups** ←
- ☒ User Groups
- ☒ Grant Security Privileges
- ☒ Manage Connected Users

Batch Commands

- ☒ Change a Problem Owner
- ☒ Change a Problem Originator
- ☒ Change a Problem Module
- ☒ Change a Problem Assigned To
- ☒ Change a Problem Version Open
- ☒ Change a Problem Module Owner

Administration menu

The Privacy Group screen appears.

- Click on the **Add** button.

PRIVACY_GROUP

Add a new privacy group

Privacy Group Name	Title To Display On All Screens
TEST	Test
GROUP_XYZ	group xyz

2 records selected

Return

Privacy Group screen

The Add A Privacy Group screen appears.

Add a privacy group

Fixed database name: division_a

Title to display on screens: Division A

Update Cancel

Add Privacy Group screen

- Type in the name of your new Privacy Group that is relevant to the group name as a title (I). Database names have no spaces, and are composed of letters, numbers, and the characters -, _, and +.
- Type in a title name (II).
- Click the **Update** button (III).

The System Administrator may add users to Privacy Groups by clicking the checkbox beside the desired Privacy Group name on the Add New user screen, or the Update User Details screen under User Accounts on the Administration menu.

Privacy Groups

☐ group xyz

☐ Test

Update Cancel

Home phone

Cell phone

Fax

Pager

Add new user screen

BUSINESS RULES & WORKFLOW

In addition to user and security management, ExtraView allows the System Administrator to manage workflow by creating and enforcing state change rules and business rules.

Creating State Change Rules

State Change Rules allow the System Administrator to control the process by which the statuses of issues can be altered. To conform to your company's workflow, ExtraView can create State Change Rules based on three different workflow formats:

- I. **Default Format:** All users must follow the same rules for all different product, projects, categories, modules etc.
- II. **User Group Format:** Different state change rules apply to different user groups within your company.
- III. **Product Format:** Certain products can be changed to statuses that are different from those that apply to other products.

Create State Change Rules

1. From the Administration menu, click on **Default Values & Behavior Settings**.
2. Click the **Edit** button next to the default value entitled **ENFORCE_STATE_CHANGE_RULES**.
3. Change the value from "No" to "Yes," and click the **Update** button (if the value is already "Yes" then leave it so).

Note: For the next step, you will have to decide which workflow is best for your company. You are able to choose between DEFAULT, PRODUCT and USERGROUP (as discussed above). The most common choices are DEFAULT and USERGROUP. PRODUCT will only be used if you want to restrict certain products from being set to certain statuses.

4. From Default Values & Behavior Settings, click the **Edit** button next to the **SEPARATE_WORK_FLOW** default value.
5. Change the value to DEFAULT, USERGROUP or PRODUCT and press the **Update** button.

Customize State Change Rules

From the Administration menu, under Business Rules & Workflow, click on **State Change Rules**.

Business Rules & Workflow

☒ State Change Rules

Based on your choice of workflow format, one of the following three screens will appear (the screens may look slightly different than what is represented here, but the functionality is identical).

State Change Rules

Check the boxes of the allowable state changes you want to allow users to make. Remember that all the rules can be switched off or on together in the Default Values & Behavior Settings screen.

From Status	To Status				
	Closed	Fixed	Open	Pending	Unassigned
Closed	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Fixed	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Open	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Pending	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Unassigned	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Fig.1-State Change screen based on selection of DEFAULT

State Change Rules

Check the boxes of the allowable state changes you want to allow users to make. Remember that all the rules can be switched off or on together in the Default Values & Behavior Settings screen.

Select User Group Administrator

From Status	To Status				
	Closed	Fixed	Open	Pending	Unassigned
Closed	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Fixed	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Open	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Pending	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Unassigned	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Fig.2-State Change Rules screen based on selection of USERGROUP

State Change Rules

Check the boxes of the allowable state changes you want to allow users to make. Remember that all the rules can be switched off or on together in the Default Values & Behavior Settings screen.

Select Product: DevTest

		To Status				
		Closed	Fixed	Open	Pending	Unassigned
From Status	Closed		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	Fixed	<input checked="" type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	Open	<input checked="" type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>
	Pending	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>
	Unassigned	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Fig.3-State Change Rules screen based on selection of PRODUCT

Enforce State Change Rules

- For PRODUCT or USERGROUP workflows, select a PRODUCT or USERGROUP from the dropdown list (there is no need to perform these steps for DEFAULT).
- Start by clicking the check boxes for each **From Status** (left-hand side) to the **To Status** (right-hand side).
 - Fig. 1 shows a scenario where a user is able to change the status of an issue from Fixed to Closed, but not from Open to Closed.
 - Fig. 2 shows a scenario where an Administrator is able to change the status of an issue from Fixed, Open, and Pending to Closed, but not from Unassigned to Closed.
 - Fig. 3 shows a scenario where the product DEV TEST can be changed only to a status of Closed, and nothing else. This may indicate that the particular company is no longer developing the product DEV TEST.
- After you are satisfied with the **From** and **To** values, click the **Update** button. Select another **Product** or **User Group** from the list, and follow steps 1 & 2. Do this for each **Product** or **User Group**.

After your final update, your State Change Rules are fully implemented. You can now test out the functionality by editing an issue and trying to change the **From Status** to a **To Status** that you did not specify.

Customize Business Rules

ExtraView has the ability to apply business rules based on your particular business processes. Customized Business Rules can include: forcing users to add *Release Notes* when they close an issue; making sure that *Comments* are added when a

disposition is *Fixed*; almost anything else you can imagine. The screenshot below is an example of a Customized Business Rule.

Rule 2

This business rule is applied when updating problems only. If you check a disposition then new comments must be added whenever the disposition changes to the values displayed below:

- ☐ Cannot Duplicate
- ☐ Deferred
- ☐ Duplicate
- ☐ Fixed
- ☐ Invalid
- ☐ Need more info
- ☐ None
- ☐ Not a bug
- ☐ User error

Sample Customized Business Rule Implementation screen

Issuing Batch Commands

A Batch Command is an ExtraView feature that allows you to edit any number of records in only a few small steps. Listed below are the standard Batch Commands, although your system may have different Batch Commands or alternative versions of the existing ones.

Batch Commands

- ☒ Change a Problem Originator
- ☒ Change a Problem Version Open
- ☒ Change a Problem Component
- ☒ Change a Problem Owner
- ☒ Change a Problem Module

Batch Command Screen

Issue Batch Commands

We will use Owner for purposes of example, but all batch commands will work in a comparable manner.

1. From the Administration menu, click on **Change a Problem Owner**.

A screen similar to the one below appears.

Change the Problem Owner

This command will re-assign all the problems that are currently owned by the person in the first list box, and will change the ownership to the person in the second list box. Only problems with the states selected will be processed. Note that you can only change the ownership to a person who has an activated account.

Current Owner ?

Change to ?

Status

- ☒ None
- ☒ Closed
- ☒ Fixed
- ☒ Open
- ☒ Pending
- ☒ Unassigned

Change Problem Owner Batch Command Screen

2. Select the **Current Owner** from the dropdown list box, or in this case, from the pop up text box. This is the name of the person whom you no longer want to be associated with the issues.

Current Owner ?

3. Select the **New Owner** from the dropdown list box, or in this case from the pop up text box. This is the person who will be the new **Owner** of the issues.

Change to ?

4. Decide which statuses you would like transfer to the new **Owner**, and then check the appropriate boxes.

Status

- ☒ None
- ☒ Closed
- ☒ Fixed
- ☒ Open
- ☒ Pending
- ☒ Unassigned

5. Click the **Execute** button after your statuses have been selected.

The changes may affect a large number of records in your database; the screen below gives you a final opportunity to cancel or to continue with the change.

Change the Problem Owner

If you go ahead and execute this command, the ownership of 6 problems will be reassigned from robbie.lloyd to rob.lloyd.

You will not be able to undo this change. Press the Execute button to continue, or the Cancel button to quit.

Batch Command Confirmation Screen

6. Click the **Execute** button to proceed with the batch change. Click **Cancel** if you do not want to follow through with these modifications.

EMAIL FEATURES

ExtraView offers a number of email features that designed to facilitate maximum flexibility in managing inter-group, cross-functional, and business-to-customer communication. At the highest level, this section will be divided into the following three heads: Email Notification, Email Generation, and Interest Lists.

As a general default, ExtraView automatically sends email to the person who Originated an issue, the person who is Assigned to the issue, and the person who is selected as the Owner of the issue. Listed below are the email functions that may be defined by the Administrator and the user. The main email functions include:

Administrator-Defined Features

- Turn System-Wide Email On or Off
- Disable Email Generation for User Groups
- Enable or Disable Email to External Users
- Assign Module Owner
- Set Product Email Address
- Customize Email Subject Line
- Show Recipients at Bottom of Email

User-Defined Features

- Notify of Own Updates
- Select Email Format
- Disable Automatic Email Generation
- CC Email

Turning System-Wide Email On or Off

1. From Administration menu, click on **Default Values and Behavior Settings**.
2. Scroll down until you see EMAIL_NOTIFICATION.

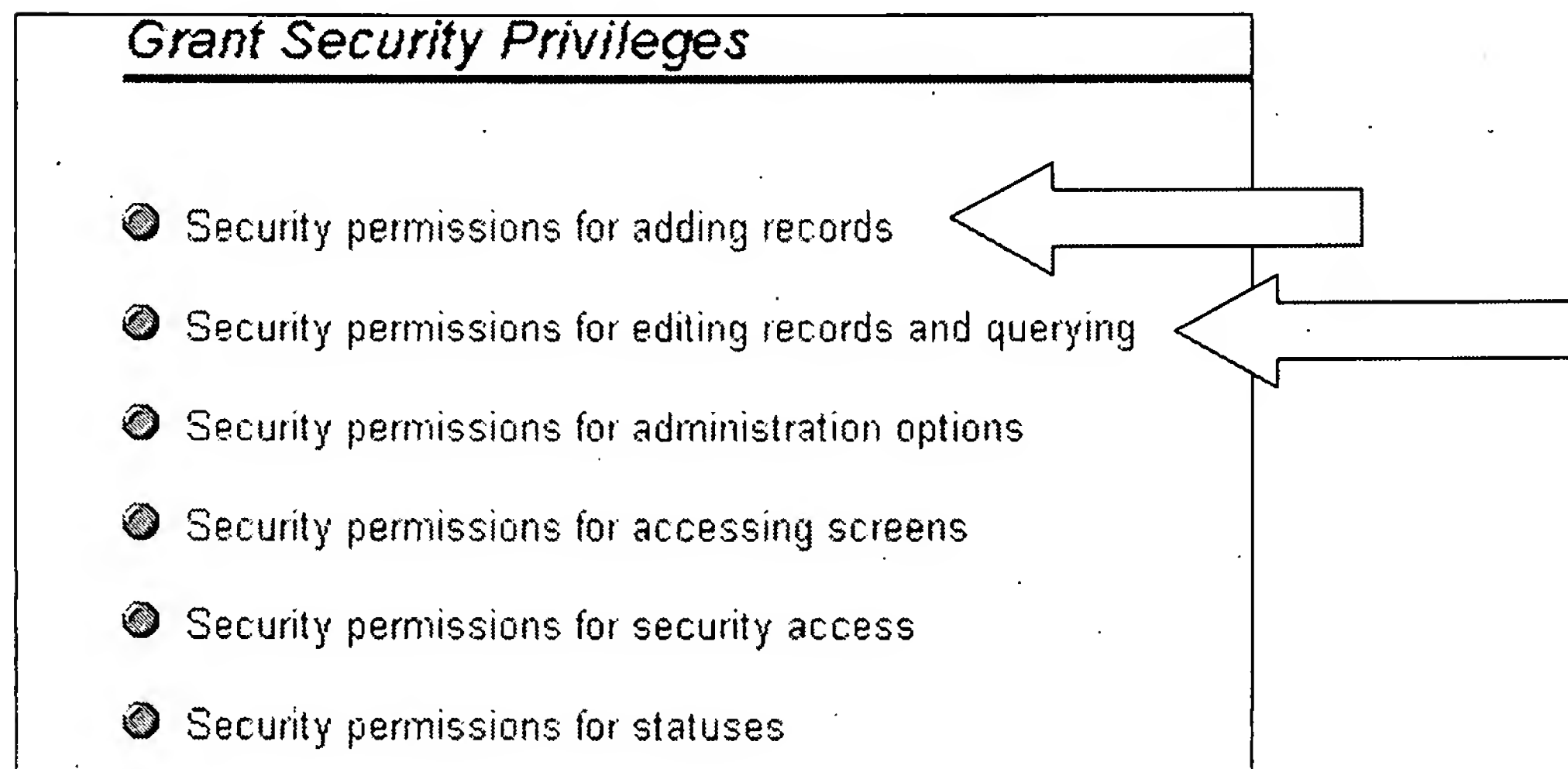
3. Click **Edit** and change the value to "Yes", in order to turn on Email Notification, or set the value to "No" to turn off **Email Notification** system-wide.
4. Click the **Update** button.

Disable Automatic Email Generation for User Groups

The System Administrator may turn off the Generate Email field for particular User Groups, so that users belonging to these groups will not have the option to disable automatic email generation.

1. From the Administration menu, click on **Grant Security Privileges**.

The following screen will appear:



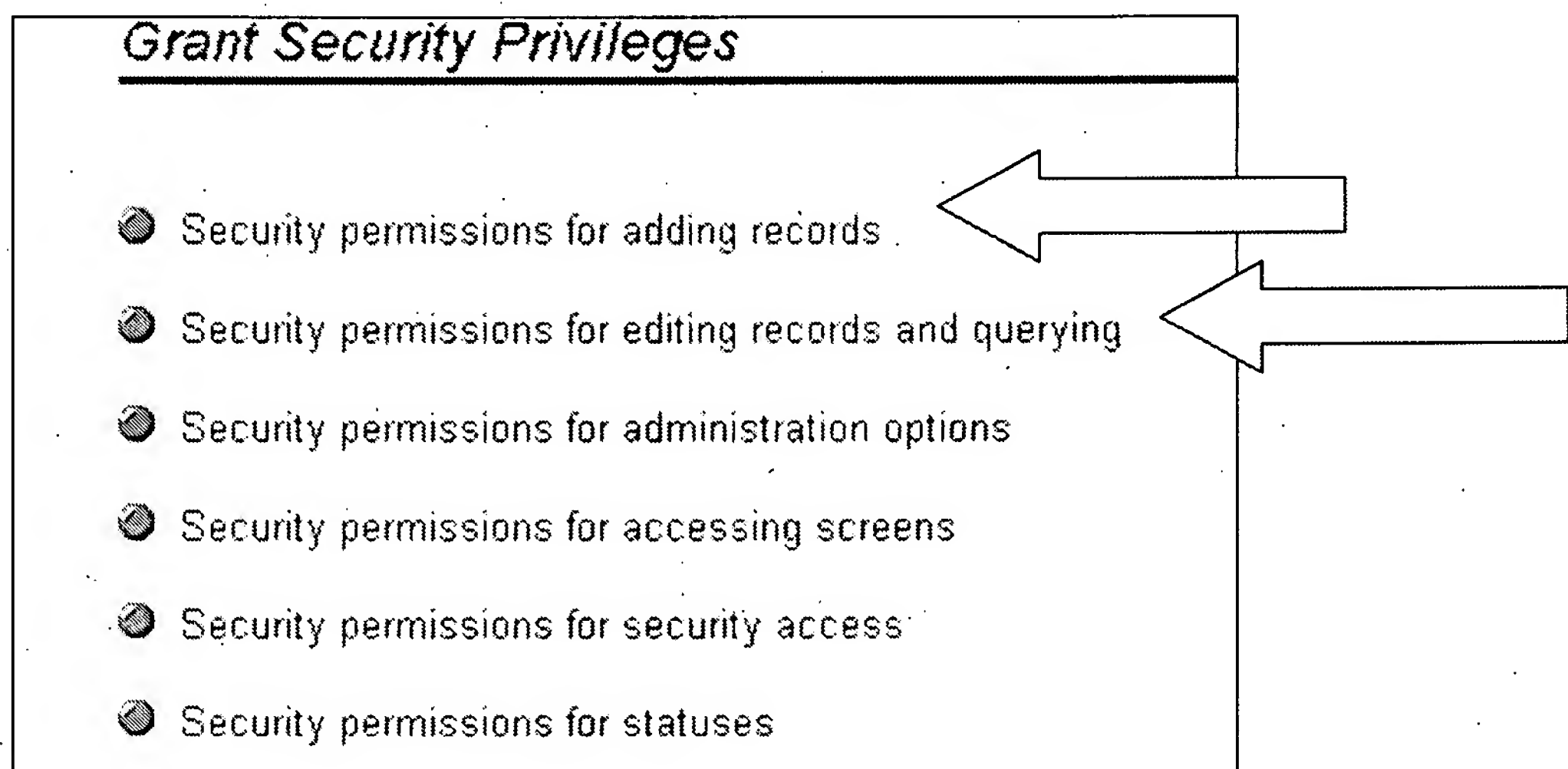
Grant Security Privileges Screen

2. To turn the Generate Email field option off for the Add Problem screen, click **Security Privileges for Adding Records**.
3. To turn the Generate Email option off for the Edit Problem screen, click **Security Permissions for Editing Records and Querying**.
4. Scroll down until you see PR_ADD_PROBLEM.EMAIL_SWITCH (this is for the Add screen) and PR_RESOLUTION.EMAIL_SWITCH (this is for the Edit screen).

To turn this field off for a specific User Group, uncheck the **Read** and **Write** boxes (for that group only) and click the **Update** button at the bottom of the screen.

Disable Include Guests for Screens or User Groups

1. From the Administration menu, click on **Default Values and Behavior Settings**.
2. Verify that IGNORE_USER_GROUP is equal to the user group you are using for external users. This is normally Guest or Customer.
3. If you make a change, be sure to click the **Update** button.
4. From the Administration menu, click on the option that says **Grant Security Privileges**.
5. To turn the External Email option off for the Add Problem screen, click **Security Permissions for Adding Records**.
6. To turn the External Email option off for the Edit Problem screen, click **Security Permissions for Editing Records and Querying**.



Grant Security Privileges Screen

7. Scroll down until you see PR_ADD_PROBLEM.EMAIL_CUSTOMER (this is for the Add screen) and PR_RESOLUTION.EMAIL_CUSTOMER (this is for the Edit screen).

Customer Email		Read	Write	Read	Write	Read	Write	Read	Write
Send Email to Customer		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
PR_ADD_PROBLEM.EMAIL_CUSTOMER									

Security Privileges for Emailing External Users

8. To turn this field on or off for a specific user group, check or uncheck the appropriate **Read** and **Write** boxes (for that group only) and click the **Update** button at the bottom of the screen.

Assign Module Owners

The System Administrator may assign Module Owners, such that state changes to given modules produce automatic email notification for designated owners.

1. From the Administration menu, click **Module Names**.
2. Locate the Module to which you would like to assign an Owner, and click the associated **Edit** button.
3. Scroll through the list of available users or else select the owner from the pop up text box.
4. Click the **Update** button.

Set Product Email Address

The System Administrator may set an email address for specific *products*, so that ExtraView users associated with that *product* will receive automatic email notification about product-related issues.

Customize Email Subject Line

ExtraView gives the System Administrator the ability to customize the subject line of incoming emails.

1. From the Administration menu, click on **Default Values and Behavior Settings**.
2. Scroll down until you see EMAIL_SUBJECT_TEMPLATE and click the associated Edit button.
3. Type in the names of the fields you would like to display. Your custom email subject line can include any field from your ExtraView installation. These values are generated dynamically, based on the particular issue.
4. If you would like values in your email subject line you must surround them with "\$\$". Normal static values can be typed in. See the example below:

Sample Text	Subject Line Output
\$\$ID\$\$ - \$\$SHORT_DESC\$\$	12345 – Problem with List Entries
\$\$ASSIGNED_TO\$\$ (\$\$PRODUCT_NAME\$\$; \$\$MODULE_NAME\$\$) - This is an Email	Rlloyd (Product X; Module Y) - This is an Email
Issue # \$\$ID\$\$ This is Assigned to \$\$ASSIGNED_TO\$\$	Issue # 12345 This is Assigned to Rlloyd

Notify of Own Updates

This feature gives you the option to disable automatic email to yourself on issues that you **Add** or **Edit**.

1. From the Administration menu click on the User Accounts option
2. Click the *Edit* button next to your User ID. A screen similar to the one below will appear.

Change a user's details

User ID	ROBBIE.LLOYD	Email address	<input type="text" value="rlloyd@sesame.com"/>
First name	<input type="text" value="Robbie"/>	Email format	<input type="text" value="HTML"/>
Last name	<input type="text" value="Lloyd"/>	Date format	<input type="text" value="DD-MON-YYYY hh24:mi"/>
Password	<input type="password" value="*****"/>	Drilldown Report format	<input type="text" value="Detailed Report"/>
Verify Password	<input type="password" value="*****"/>	Timezone	<input type="text" value="(GMT -4:00) Atlantic Time(US & Canada)"/>
Expire password	<input type="checkbox"/>	Notify on own updates	<input checked="" type="radio"/> Yes <input type="radio"/> No
User account	<input checked="" type="radio"/> Enabled <input type="radio"/> Disabled	Job title	<input type="text"/>
User groups	<input checked="" type="checkbox"/> Administrator <input type="checkbox"/> Development Manager		

Change a User's Details screen

Note: Users can also turn this feature "On" or "Off" by clicking on **Click to edit your personal options**, from their ExtraView Home Pages.

☒ Click to edit your personal options.

3. If you do not want to receive email notification each time you update an issue, set the value in **Notify of own updates** to "No".

☐ Yes ☒ No

4. Click the **Update** button.

Select Email Format

Choose to display incoming email in one of three formats: HTML, Plain Text (full) and Plain Text (brief). Plain Text (full) displays the entire email, while Plain Text (brief) shows just a few lines, so that the issue may be recognized.

1. From the Administration menu, click on the **User Accounts** option
2. Click the **Edit** button next to your User ID.

A screen similar to the one below appears.

The screenshot shows a web form titled "Change a user's details" for user ROBBIE.LLOYD. The form is divided into two columns. The left column contains fields for First name (Robbie), Last name (Lloyd), Password (masked with asterisks), and Verify Password (masked with asterisks). There is an "Expire password" checkbox and radio buttons for "User account" (Enabled/Disabled). The "User groups" section has checkboxes for "Administrator" (checked) and "Development Manager". The right column contains fields for Email address (r1loyd@sesame.com), Email format (HTML), Date format (DD-MON-YYYY hh24:mi), Drilldown Report format (Detailed Report), Timezone ((GMT -4:00) Atlantic Time(US & Canada)), Notify on own updates (Yes/No), and a Job title field.

Change User Details screen

Note: Users can also turn this feature "On" or "Off" by clicking on **Click to edit your personal options** from their ExtraView Home Pages.

Click to edit your personal options.

3. The default email format is HTML. If you would like to see your email notifications in another format, select the desired format from the **Email format** list.

A large white arrow points to a dropdown menu for "Email format". The menu is open, showing "HTML" as the selected option. Below it, the other two options are listed: "Plain text (full)" and "Plain text (brief)". The "Date format" and "Drilldown Report" labels are visible to the left of the dropdown.

Email Format Option

Below are examples of the different email formats:

Subject: DEV - 12231
 Date: Fri, 15 Jun 2001 16:53:09 -0700 (PDT)
 From: "ExtraView" <extraview.qa@sesame.com>
 To: "Robbie Lloyd" <rlloyd@sesame.com>

12231	12231	test		
Product	Owner	Created	Last Modified	
NetOp	udale malabanan	08-MAR-2001	15-JUN-2001 16:51	
Category	Assigned To	View	ABSD	Changed By
Hardware	rob lloyd	Private		ROB.LLOYD
Severity	Priority			
	m2			
Module				
PM				
Component	Platform	OS	Clarify ID	Customer
	800 TDM			
Test Case ID	Test Case Location			
Version Open	Status	Version Closed	Disposition	
1	Closed			
Description:				
test				
Comments:				
Workarounds:				
Release Notes:				
Attachments:				

cc: Michelle Medina, rob lloyd, Robbie Lloyd.

HTML Email

Subject: Open [#19419]: Category doesn't seem tied to Product anymore
 Date: Wed, 2 May 2001 13:57:53 -0700 (PDT)
 From: "ExtraView" <extraview-user@customer.com>
 To: <support@sesame.com>

ExtraView Notification for Bug # 19419

*Synopsis: Category doesn't seem tied to Product anymore

Link: http://www.extraview.net/extraviewsql/SE_Signon.FrameSet?p_case_id=19419

*Product: Dev Tools
 *Originator: rlloyd
 *Owner: rlloyd
 *Changed By: dwong
 *Priority: 1
 *Severity: 1
 *View: Private
 *Last Modified: 02-MAY-2001 13:57
 *Category: Software
 Alt ID:
 Created: 02-MAY-01

*Module: ExtraView
 Component:
 *Platform: all
 OS:
 Clarify ID:
 Test Case ID:
 Test Case Location:
 Problem Reproducible in SOA:

*Version Open: 3.1.2.1
 *Status: Open
 Version Closed:
 Disposition:

*Description:
 If you select Product=EV, Category=Software, you will see two modules called "EV". Go into Admin...Modules and one is Hardware, one is Software.

Comments:

ExtraView - Copyright Sesame Technology 1999, 2000. All rights reserved.

Plain Text (Full)


```

Subject: Open [#19419]: Category doesn't seem tied to Product anymore
Date: Wed, 2 May 2001 13:57:53 -0700 (PDT)
From: "ExtraView" <extraview-user@customer.com>
To: <support@sesame.com>

ExtraView Notification for Bug # 19419

*Synopsis: Category doesn't seem tied to Product anymore

Link: http://www.extraview.net/extraviewsql/SE_Signon.FrameSet?p_case_id=19419

*Product: Dev Tools
*Originator: rlloyd
*Owner: rlloyd
*Priority: 1
Severity:


```

ExtraView - Copyright Sesame Technology 1999, 2000. All rights reserved.

Plain Text (Brief)

Disable Automatic Email Generation

Each time a user Adds or Edits an issue, he or she has the opportunity to halt all email generation by un-checking the **Generate Email** checkbox at the bottom of the Add and Edit screens.

Attachments	Description	File Name
	<input checked="" type="checkbox"/> Generate Email <input type="checkbox"/> Include external users	CC Email <input type="text"/>
<div> <input type="button" value="Update"/> <input type="button" value="Delete"/> <input type="button" value="Interest List"/> <input type="button" value="History"/> <input type="button" value="Return"/> </div>		

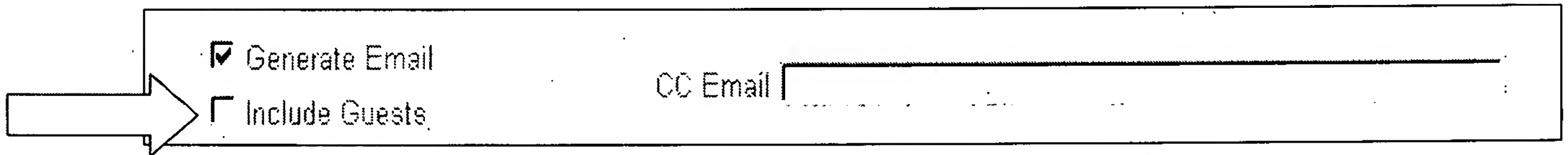
Email Portion of Edit Screen

Note: the Edit screen can be accessed by drilling down on your Home Screen folder, from the Add Problem Summary Screen, from email notification, as well as from various reports.

Disable Generate Email to External Users

Situations often arise where a customer may need to enter an issue, but you may not want the customer to see all of the different state changes that the problem issue goes through. When you **Add** or **Update** an issue, you have the option of halting email to external users.

1. At the bottom of the **Add** and **Edit** screen you will see something similar to the screenshot below. The default setting is not to send email to external users.



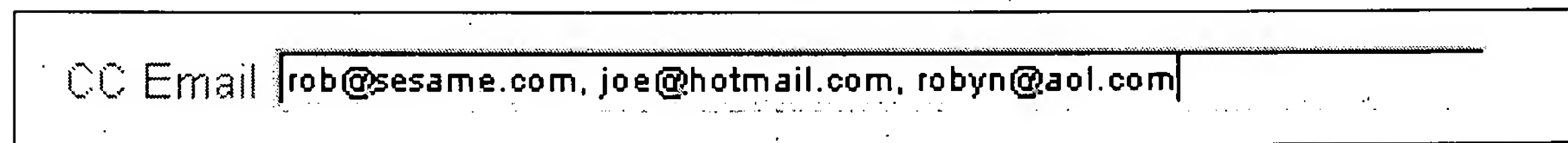
A screenshot of a form interface. On the left, a large white arrow points to the right. The form contains two checkboxes: 'Generate Email' which is checked, and 'Include Guests' which is unchecked. To the right of these checkboxes is a text input field labeled 'CC Email'.

2. If you would like your external users to receive an email, click the **Include Guests** check box and continue to update the issue.

Enable CC Email

This feature gives users the opportunity to send a one-time email to people who are not directly associated with a particular issue.

1. To utilize the CC Email functionality, type email addresses (separated by commas or semi colons) into the CC Email field prior to updating the issue.



A screenshot of a form field labeled 'CC Email'. The field contains the text: 'rob@sesame.com, joe@hotmail.com, robyn@aol.com'.

CC Email Field

Email Interest Lists

ExtraView allows users to subscribe to interest lists for nearly any field in the system. An Interest list allows users to receive emails about particular Metadata in the system. For example, you may want to notify a certain user each time an issue goes to a *Severity of Critical* or a *Status of Open*. Interest lists can be created for *Products, Modules, Statuses, Priorities, Resolutions, Severities* and *User Defined Fields*.

The first example below will illustrate the addition and management of interest list members for normal Configuration items (*Product, Module, Priority* etc.). The second example will be illustrative for User Defined Fields (fields specific to your installation).

Create Normal Configuration Item Interest Lists

Normal Configuration items include items such as *Product, Module* and *Priority*. For this example, we will use *Product (PRODUCT_NAME)*; adding Interest List members for the other configuration items will be the same.

1. From the Administration menu, click **Data Dictionary** under Installation & Setup.
2. Scroll down until you see *Product* (PRODUCT_NAME), and click the *Edit* button next to this value.
3. Scroll down until you see the following:

Enable interest list on this field <input checked="" type="radio"/> Yes <input type="radio"/> No
--

Option to Enable Interest Lists

4. Check the **Yes** radio button.
5. Click the **Update** button at the bottom of the screen.
6. Click **Administration** and scroll down to near the bottom of the page.
7. Click on **Products**.
8. Locate the *Product* or *Products* to which you would like to add Interest List members, and click the associated **Edit** button.

The following screen will appear:

<i>Change Product details</i>	
Fixed database name NETTRANSACTIONS	
Title to display on screens	<input type="text" value="Net Transactions"/>
E-Mail Address	<input type="text"/>
Active	<input checked="" type="radio"/> Yes <input type="radio"/> No
<input type="button" value="Update"/> <input type="button" value="Delete"/> <input type="button" value="Cancel"/> <input type="button" value="Product Interest List"/>	

Change Product Details Page

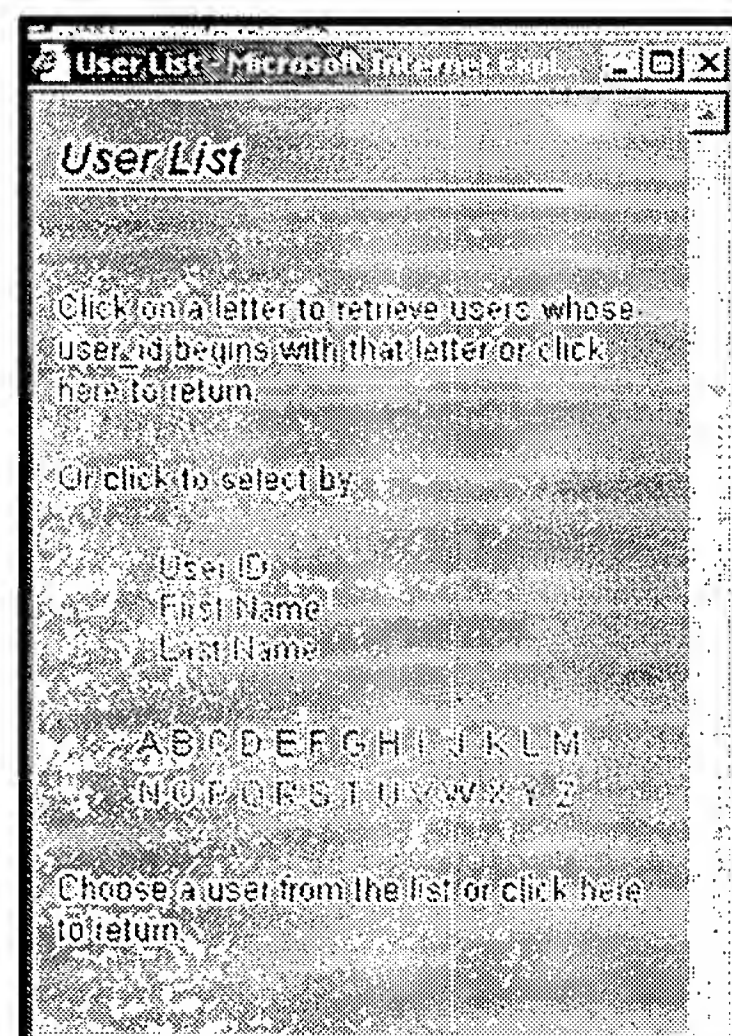
9. Click the **Product Interest List** button.
10. Click the **Add a new member to the Interest List** button and the following screen will appear:

Add a new member to the interest list for Tech Pubs

Select the user to add to the interest list

Add a New Member to Product Interest List screen

11. From the screen above, you can select the member of your organization you would like to add to the *Tech Pubs* (product for this example) interest list by selecting the question mark. A screen like the one below will appear, in which users can be chosen alphabetically, by user ID, First Name, or Last Name.



12. Continue to add users as desired by repeating step 11.

Create User-Defined Field Interest Lists

All Interest List members will be automatically sent email each time this product is involved with a problem issue. The same process can be used for *Products, Modules, Statuses, Priorities, Resolutions, and Severities*.

For this example, we will use the User-Defined Field *Platform*. To add members to UDF Interest Lists:

1. From Administration menu, click **Data Dictionary**.

2. Scroll down until you see the *PLATFORM* UDF.
3. Click the associated *Edit* button.
4. Scroll down until you see the following:

Enable interest list on this field <input checked="" type="radio"/> Yes <input type="radio"/> No

Option to Enable Interest Lists

5. Check the **Yes** radio button.
6. Click the **Update** button at the bottom of the screen.
7. From the Administration menu, click on **User Defined Fields**.
8. Click the **List** button next to *Platform*.
9. Click the **Edit** button next to the particular *Platform* to which you want to add interest list members.
10. Click the **Add a new member to the Interest List** button.

The following screen will appear:

Add a new member to the interest list for 1000	
Select the user to add to the interest list	<input type="text"/> ?
<input type="button" value="Update"/>	<input type="button" value="Cancel"/>

Add a new member to the UDF Interest List screen

11. Select the member of your organization whom you would like to add to the *1000 Platform* Interest List.
12. Continue to add users to the **1000 Interest List** as desired by repeating step 11.

The same process can be used for any other list **UDFs** in your system.

Edit Personal Email Interest Lists

A Personal Interest List can be maintained to allow someone other than the owner of a problem to track its progress. When a problem is placed on a Personal Interest List, the user will automatically receive an email each time a change is made to a problem in the database.

Emails are automatically generated to the owner of the *problem* and the assigned person of the new problem entered. If you are changing the state or other information related to a *problem*, then email is sent to the owner, the originator and the person assigned to the *issue*. If any of these values are the same, only one email will be generated per person. In addition, users can subscribe to an interest list for any problem.

1. Click the **Interest List** button at the bottom of any given Edit screen.
2. Click the **Add a new member to the interest list** button to add a person to the interest list.
3. Select the name of the user from the list that is presented.
4. Click the **Update** button.

You can edit your inclusion on an interest list if you belong to one, but this does not apply to Product, Module or other specific field lists.

Remove User ID from Interest List

1. From your user Home Page, press the **Click to edit your personal email interest list** button.

Problems you originated:

Status	View	Count	P0	P1	P2	Other
Unassigned						

Use this quick search for simple queries. For more complex searching use the Search/Report menu button.

Status	<input type="text" value="Any"/>	Priority	<input type="text" value="Any"/>
Assigned To	<input type="text" value=""/>	Owner	<input type="text" value=""/>
Product	<input type="text" value="Any"/>	From Date	<input type="text" value=""/>
Category	<input type="text" value="Any"/>	To Date	<input type="text" value=""/>
Module	<input type="text" value="Any"/>	Date created	<input type="text" value=""/>
Version Open	<input type="text" value=""/>	Last Modified	<input type="text" value=""/>

Sort By	<input checked="" type="radio"/> Priority	<input checked="" type="radio"/> Ascending
	<input type="radio"/> Bug ID	<input type="radio"/> Descending
	<input type="radio"/> Date Modified	

Records per Page

☒ Click to change your current user role.

☒ Click to edit your personal options.



☒ **Click to edit your personal email interest list**

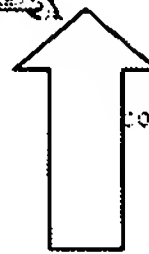
Home Page screen

2. Click the **Edit** icon beside the issue number from which you want to remove yourself.

Email Interest List for Darrin Wong

If you would like to be added to or deleted from other interest lists, please contact your system administrator.

	Bug #	Status	Title
	12335	Unassigned	Test for state change rules
	12355	Unassigned	calendar


 2 records selected

Personal E-mail Interest List screen

2. Click the **Delete** button and you will be removed from the interest list.

Delete an interest list entry for dwong

Delete interest list entry for Bug # 12518
Status: Unassigned
Title: This is a problem



Delete Interest List Entry screen

ExtraView Help

The ExtraView application includes a comprehensive HTML-based help system that you can access at any time by clicking the **Help** button on the navigator frame. In addition, many tool tips and context-sensitive links are defined throughout the application.

When you place the mouse cursor over a screen label that has a tool tip, a small window will appear next to your mouse cursor with a definition of what this label does. These labels allow you to define help tips for your users. If you press the mouse button over a label, you will be taken to a specific page within the help system. If you do not have a specific page defined within the screen name Administrative section, you will be taken to the Help Index page. This page consists of links to detailed information about the system.

ExtraView Help Index

This is the main Help Index for the ExtraView problem tracking and resolution system. From here you can navigate to any help topic. Help windows may be left open at all times that you are working on problem reports within ExtraView.

Help Topic List

- Before You Get Started
- Adding a New Problem
- Editing an Existing Problem
- Search/Report
 - Reporting & Searching Capabilities
 - Searching the Problem Database
- Security
 - Create and Maintain User Accounts
 - System Security Keys
 - User Groups
 - Grant Security Privileges to User Groups
 - Allowing User Groups to see only certain Statuses
- Reviewing Problem History
- Administration
 - Allowed Values
 - Batch Commands
 - Categories
 - Changing Personal Details
 - Data Dictionary
 - Default Values & Behavior Settings
 - Enclosure Types
 - Installation Setup
 - Module Names
 - Module Types
 - Priorities

ExtraView Help screen

REF: U.S. PATENT
APPN. SER. NO.
10/767,511
EXHIBIT H

From eternity to 16 seconds.txt
From: Rick Banister [rick@sesame.com]
Sent: Wednesday, May 08, 2002 4:32 PM
To: Tom Strzemieczny; Maria Scharin; Carl Koppel; Michael Stebbins;
Steve Hoydic
Cc: dev@sesame.com
Subject: From eternity to 16 seconds

ONE PAGE

I've got the worse case query so far down to 14 second, which is to query on 5 problem fields, 1 udf, and sort by a UDF. The EXPLAIN PLAN shows ridiculously high cost, but the actual performance is very good, even with the SGA cache cleared by restarting the database.

There are two items, a DDL change to bring the values into the index for an index-only search, and changing the query to use FIRST_ROWS so it doesn't try a full index query of the PROBLEM_UDF table:

```
DROP INDEX IX_PROBLEM_UDF1;  
CREATE INDEX IX_PROBLEM_UDF1  
  ON PROBLEM_UDF (problem_id, udf_id, VALUE, VALUE_NUMBER, VALUE_DATE, UDF_LIST_ID  
)  
  TABLESPACE EXTRAVIEW_IDX;  
ANALYZE INDEX IX_PROBLEM_UDF1 ESTIMATE STATISTICS SAMPLE 20 PERCENT; ANALYZE table  
PROBLEM_UDF ESTIMATE STATISTICS SAMPLE 20 PERCENT;
```

```
select /*+ ORDERED first_rows  
  INDEX(PROBLEM1 IX_PROBLEM5)  
  INDEX(PROBLEM2 IX_PROBLEM6)  
  INDEX(PROBLEM3 IX_PROBLEM7)  
  INDEX(PROBLEM4 IX_PROBLEM8)  
  INDEX(PROBLEM5 IX_PROBLEM2)  
  index(pudf ix_problem_udf1)  
*/ PROBLEM1.ID  
from PROBLEM PROBLEM1, PROBLEM PROBLEM2, PROBLEM PROBLEM3 , PROBLEM  
PROBLEM4 , PROBLEM PROBLEM5, problem_udf pudf, udf_list where PROBLEM1.PRIORITY in  
( 'PRIORITY 2' ) and PROBLEM2.SEVERITY_LEVEL in ( 'MAJOR' ) and PROBLEM3.STATUS in  
( 'OPEN' ) and PROBLEM4.PRODUCT_NAME in ( 'EVJAVA' ) and PROBLEM5.ASSIGNED_TO in  
( 'AIMEN' ) and PROBLEM2.ID = PROBLEM1.ID and PROBLEM3.ID = PROBLEM2.ID and  
PROBLEM4.ID = PROBLEM3.ID and PROBLEM5.ID = PROBLEM4.ID and pudf.problem_id(+) =  
problem5.id and nvl(pudf.udf_id(+),88) = 88 and udf_list.udf_list_id(+) =  
pudf.udf_list_id and exists  
  (select 1  
  from problem_udf udf1  
  where udf1.problem_id = PROBLEM5.id  
  and udf1.udf_id = 88  
  and udf1.udf_list_id = 6231)  
ORDER BY udf_list.title DESC
```

REF: U9 PAT PPN.
SER. NO.

10/767,500

EXHIBIT G

ONE PAGE

mo betta SQL statement.txt

From: robert lange [rlange@sesame.com]
Sent: Thursday, October 10, 2002 11:59 AM
To: Dev@Sesame. Com
Cc: LeeAnn Pultz; Maria Scharin
Subject: mo betta SQL statement

I ran into some surprising results tuning a query that I want to pass on to anyone writing SQL -- at least for the Oracle platform. The change involved replacing two joins in the where clause with "pseudo tables" in the where clause. The first change decreased resources by a factor of 10,000 in the explain plan, but the resulting query still took 37 what was especially surprising is that in the final change, even though the explain plan didn't show significant difference, the execution speed changed dramatically: by a factor of 40 again.

the basic query involved reporting, where you have a list of ids, and the problem udf table. the first pass on the query looked like this (simplified for clarity):

```
select p.id, p2.id
from problem p, problem p2, problem_udf pu, udf u where p.id in (?, ?, ?, ?, ?, ?, ?, ?)
and pu.udf_id = u.udf_id and u.name = '?'
and p2.id in (select problem_id from problem_udf
              where udf_id = u.udf_id and u.value = p.id) ;
```

the key was to rewrite the code to put some of the where clause into the from clause. step 1 was:

```
select p.id, p2.id
from (select id from problem where id in (?, ?, ?, ?, ?, ?, ?, ?)) p,
     problem p2, problem_udf pu, udf u
where p.id in (?, ?, ?, ?, ?, ?, ?, ?)
and pu.udf_id = u.udf_id and u.name = '?'
and p2.id in (select problem_id from problem_udf
              where udf_id = u.udf_id and u.value = p.id) ;
```

this reduced resources per the explain plan by a factor of 10,000 or so, but the query still took 40 seconds to run for 5 ids. the second step was to apply the same strategy to eliminate the joins on the udf table:

```
select p.id, p2.id
from (select id from problem where id in (?, ?, ?, ?, ?, ?, ?, ?)) p,
     problem p2,
     (select problem_id, value from udf, problem_udf
      where udf.name = ? and problem_udf.udf_id = udf.udf_id) pu where pu.value =
p.id and p2.id = pu.problem_id ;
```

this version did not look different in the explain plan, but the execution time dropped to under a second.

I'm not sure that this is always more efficient, and it may have to do with the fact that the actual query was quite a bit more complex, but the approach clearly can have dramatic results.

enjoy,

robert